KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
KOMPIUTERIŲ TINKLŲ KATEDRA

Vytautas Leščiauskas

# SDH tinklo resursų įvertinimas ir optimizavimas

Magistro darbas

Darbo vadovas
doc. R. Plėštys

Kaunas, 2004

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
KOMPIUTERIŲ TINKLŲ KATEDRA

TVIRTINU
Katedros vedėjas
(parašas) doc. R. Plėštys
2004 05

# SDH tinklo resursų įvertinimas ir optimizavimas

Magistro darbas

Kalbos konsultantė
Lietuvių kalbos katedros lektorė
(parašas) dr. Jurgita Mikelionienė
2004 05

Recenzentas

(parašas) doc. K. Plukas
2004 05

Vadovas

(parašas) doc. R. Plėštys
2004 05

Atliko
IFM-8/1 gr. stud.
(parašas) V.Leščiauskas
2004 05

Kaunas, 2004

# Turinys

# 1 ĮVADAS

Šiuo metu tinklų operatorių naudojama programinė įranga neužtikrina automatinio maršrutų generavimo tarp pasirinktų mazgų. Dėl to operatoriai sugaišta daug laiko projektuodami tinklus rankiniu būdu. Nepaisant sugaištamo laiko vis tiek yra daromos klaidos, kurių perkraunami tinklai ir sutrinka ryšys.

Dijkstra algoritmas naudojamas ATM tinkluose. SDH įranga taip pat turi galimybę sudaryti maršrutus tarp tinklo galinių taškų. Tačiau maršrutų suradimo kriterijai dar nėra pritaikyti SDH tinklams, nes jie neįvertina srautų struktūros.

Kadangi nagrinėjamo objekto srauto vienetai yra hierarchinės struktūros, būtina atlikti šių vienetų skaitinį įvertinimą, t.y. paversti SDH srauto matavimo vienetus iš medžio tipo struktūros į realiuosius skaičius.

Užduodamas tinklo optimalumo kriterijus – tolygus santykinis apkrautumas. Tai SDH tinkluose yra gana aktualu, nes esant tokiam kriterijui tinklo resursai bus išnaudojami tolygiai ir tinklo perkrovos būtų pasiekiamos tik tuo atveju, jei trūksta duomenų srautus perduodančios aparatūros.

Realizuoti automatinį maršrutų generavimą nėra sudėtinga: čia naudojamos grafų teorijos žinios. Tačiau sprendžiant realius trumpiausio kelio tarp dviejų viršūnių paieškos uždavinius geriausiais žinomais grafų teorijos metodais dėl didelės uždavinio apimties sprendimo laikas yra nepriimtinas. Sprendimo laiką galima sutrumpinti didinant uždavinį apdorojančios sistemos darbo našumą. Tačiau jei uždavinį spręstume tarkime Belmano-Fordo algoritmu, tai jam sueikvotume $n^3$ skaičiavimų esant duomenų kiekiui n.[2] Pagerinus sistemos našumą 100 kartų, tokio uždavinio sprendimo laikas sutrumpėtų tik 4,64 karto ($\sqrt[3]{100} \approx 4{,}64$). Toks problemos sprendimo būdas taip pat netenkina, nes sprendžiančių uždavinį sistemų skaičiavimo resursai būtų išnaudojami neefektyviai. Be to tinkle didėjant mazgų skaičiui, uždavinio sprendimo laikas 100 kartų pagreitintoje sistemoje gali greitai tapti vėl koks buvo iki sistemos pagreitinimo.

Dėl šių priežasčių iškilo skaičiavimų apimties požiūriu paprastesnių algoritmų poreikis, jų sudėtingumo įvertinimas. Darbe pateikti algoritmų aprašai ir išnagrinėtas jų sudėtingumas.

Darbo tikslai yra šie:

- Nustatyti SDH resursų vienareikšmio įvertinimo metodiką;

- Pateikti grafų, vaizduojančių SDH tinklų santykinį apkrautumą, metodiką. Šiuo uždaviniu išskiriamas SDH tinklo resursų optimizavimo kriterijus – vienodas santykinis tinklo apkrautumas;
- Išnagrinėti šiuo metu naudojamų trumpiausio kelio tarp dviejų viršūnių paieškos algoritmų savybes ir įvertinti jų skaičiavimų sudėtingumą;
- Pateikti galimus šių algoritmų skaičiavimų sudėtingumo pagerinimo būdus;
- Pateikti duomenų struktūras, kurios įgalintų sumažinti skaičiavimų apimtį;
- Parinkti optimalių maršrutų skaičiavimo algoritmą, pritaikytą dideliems grafams su santykinai nedideliu vidutiniu viršūnių laipsniu;

# 2 SDH TINKLŲ APKRAUTUMO OPTIMIZAVIMO UŽDAVINIAI IR JŲ SPRENDIMO ALGORITMAI

## 2.1 Trumpas SDH aprašymas

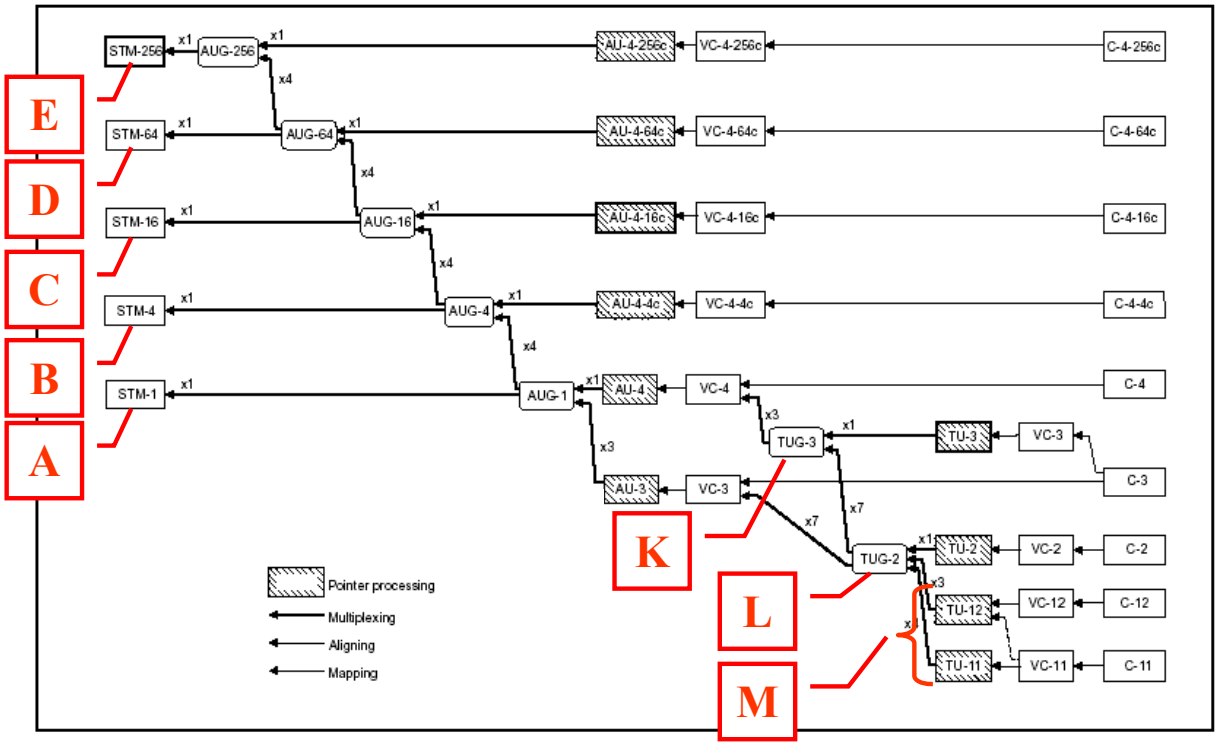
Nagrinėjant dalykinę sritį reikėtų pradėti nuo SDH tinklo sąvokų ir veikimo principų.

„STM-N srautas sudaromas pabaičiui sudėjus pavienius STM-1 srautus.

**1 lentelė SDH hierarchija**

| Bitų perdavimo sparta | SDH | Srauto spartos žymėjimas | SDH signalo talpa |
|---|---|---|---|
| 155.52 Mb/s | STM-1 | $B_{STM-1}$ | 63 E1 arba 1 E4 |
| 622.08 Mb/s | STM-4 | $B_{STM-4}$ | 252 E1 arba 4 E4 |
| 2488.32 Mb/s | STM-16 | $B_{STM-16}$ | 1008 E1 arba 16 E4 |
| 9953.28 Mb/s | STM-64 | $B_{STM-64}$ | 4032 E1 arba 64 E4 |
| 39813.12 Mb/s | STM-256 | $B_{STM-256}$ | 16128 E1 arba 256 E4 |

STM-N srauto bitų perdavimo sparta yra N kartų didesnė nei STM-1 sraute (žr. 1 lentelę). N-tojo lygmens konteineris (C-n: n=11, 12, 2, 3, 4) sutalpinamas ir į atitinkamą virtualųjį konteinerį VC-n, į kurį įeina vartotojo duomenys ir kontrolės bei valdymo informacija.




**1 pav. srautų tankinimo struktūra SDH tinkluose**

SDH tinklo mazguose tankinama remiantis G.707 rekomendacija [5], srautai juose žymimi KLM numeriais. KLM numeracijos tvarka vaizduojama 1 pav. Intakinio vieneto *TU-12* numeris tankinimo schemoje žymimas *M* raide, trijų TU grupės *TUG2* numeris žymimas *L* raide, o septynių TUG2 grupė *TUG3* – raide *K*.

**2 lentelė. Vartotojo informacijos talpinimas**

| **Srauto spartos žymėjimas** | **C-n *bitų perdavimo sparta*** | ***Žemesnės eilės virtualus konteineris* VC-n** | **Srauto spartos žymėjimas** | **VC-n *bitų perdavimo sparta*** |
|---|---|---|---|---|
| $B_{C-11}$ | *1.544 Mb/s* | *VC-11* | $B_{VC-11}$ | *1.728 Mb/s* |
| $B_{C-12}$ | *2.048 Mb/s* | *VC-12* | $B_{VC-12}$ | *2.304 Mb/s* |
| $B_{C-2}$ | *6.312 Mb/s* | *VC-2* | $B_{VC-2}$ | *6.912 Mb/s* |
| $B_{C-3}$ | *34.368 Mb/s* | *VC-3* | $B_{VC-3}$ | *48.960 Mb/s* |
| $B_{C-4}$ | *139.264 Mb/s* | *VC-4* | $B_{VC-4}$ | *150.336 Mb/s* |

Žemesnio lygmens VC-n pradžios vietą aukštesniojo lygmens VC-N konteineryje nurodo TU rodyklės, o aukštesniojo lygmens VC-N vietą cikle STM-N nurodo AU rodyklė.

Kaip buvo minėta 6psl. žemos eilės konteineris C talpinamas į virtualų konteinerį VC. Šios funkcijos esmė – naudojant užpildymo bitus sukurti standartinį VC. Toliau konteineriai VC išdėstomi į intakinius vienetus TU, kuriuose yra informacija apie VC-n pradžią. Kitu žingsniu TUG tankinamos į aukštos eilės konteinerius VC, t.y. TUG-2 tankinamas į TUG-3 arba į VC-3 ir TUG-3 tankinamas į VC-4. Suformuojami administravimo blokai AU, kurie tankinami į grupes AUG. Iš AUG sudaromi SDH pernašos srautai STM (1 pav.).

SDH tinkle tankinti naudojamų signalai ir konteineriai:

- ✓ 2 Mb/s signalas C-12 suformuojamas į VC-12, šis išdėstomas intakiniame bloke TU-12. Iš trijų TU-12 sudaroma grupė TUG-2, o iš septynių TUG-12 – grupė TUG-3, kurios trys sutankinamos į aukštesnės eilės konteinerį VC-4. Apjungimo schema būtų tokia:

C-12→ VC-12→ TU-12→TUG-2→TUG-3→VC-4;

- ✓ 34/45 Mb/s signalas C-3 suformuojamas į VC-3, šis rikiuojamas į intakinį bloką TU-3, iš jo sudaroma grupė TUG-3, kurios trys sutankinamos į aukštesnės eilės konteinerį VC-4. Apjungimo schema būtų tokia:

C-3→VC-3→TU-3→TUG-3→VC-4;

- ✓ 140 Mb/s signalas C-4 iš karto sutankinamas į aukštesnės eilės konteinerį VC-4. Apjungimo schema būtų tokia:

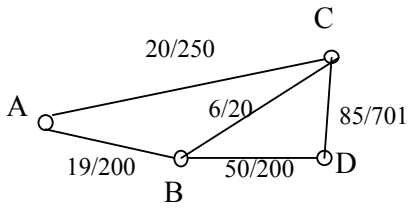
C-4→VC-4;

- ✓ Gauti VC-4 sutankinami į AUG-1. Iš vienos AUG-1 sudaromas modulis STM-1. Apjungimo schema būtų tokia:

VC-4→AU-4→AUG-1→STM-1.

Iš keturių AUG-1 sudaroma grupė AUG-4 ir modulis STM-4, iš keturių AUG-4 – grupė AUG-16 ir modulis STM-16. Taikant daugiklį 4, gaunami moduliai STM-64 bei STM-256“[5].

Ne visada yra tikslu laisvus resursus skaičiuoti TU-n lygyje. Gali susidaryti tokia situacija, kai TU-n lygyje yra santykinai didelis kiekis laisvų resursų, tačiau sukomutuoti norimo srauto nėra įmanoma. Taip nutinka abonentams atsijungiant nuo mazgo. Kiekvieną kartą atsijungus abonentui srautas, kuris buvo jam suformuotas, tampa nepanaudotas. Tačiau jei kiti to paties lygio srautai nėra atlaisvinami, tai aukštesnio lygio srautas toliau lieka panaudotas. Tokiu atveju galime gauti labai dviprasmišką situaciją (pav. 2).




**2 pav. Tinklas, kuriame gali būti neįmanomas papildomo STM-1 srauto įvedimas**

Kiekvienos briaunos svoris nurodomas santykiu m/n, čia m yra rezervuotų ir panaudotų resursų, matuojamų TU-11 kiekiu, suma, o n – instaliuotų resursų kiekis. Šiame tinkle sukomutuoti vieną STM-1 srautą tarp mazgų A ir D gali būti neįmanoma, nepaisant to, kad STM-1 srautą sudaro tik 84 svorio vienetai. Tai priklauso nuo to, kaip paskirstyti panaudoti resursai. Esant bent vienam žemesnio lygio panaudotam resursui, visi aukštesnių lygių resursai bus taip pat laikomi panaudotais. Todėl jei pav. 2 pavaizduoti panaudoti resursai yra sudėti skirtinguose žemesnio lygio srautuose, tai visi aukštesnio lygio srautai bus panaudoti. Dėl to prieš optimizavimą yra būtina arba perkomutuoti resursus taip, kad panaudoti resursai padarytų neprieinamais kaip įmanoma mažiau instaliuotų resursų. Šiose matricose resursai galėtų būti skaičiuojami tiek nagrinėjamo lygio vienetais tiek TU-n, kadangi šie svorio matai susiję per proporcingumo koeficientą.

## 2.2 Tinklus aprašantys grafai

Tinklo mazgų grafo viršūnėse yra tinklo mazgai, o briaunos – mazgus jungiantys virtualūs konteineriai VC-11. Instaliuotus resursus pateikiame gretimumo matrica:

$$\{r_{ij} | i=\overline{1,n}\ ;\ j=\overline{1,n}\} \qquad (1)$$

čia i ir j – mazgų numeriai, n – grafo mazgų skaičius, o matricos elementas $r_{ij}$ nusako instaliuotą pralaidumo resursą tarp viršūnių i ir j.

Panaudotus resursus pateikiame gretimumo matrica:

$$\{p_{ij} | i=\overline{1,n}\ ;\ j=\overline{1,n}\} \qquad (2)$$

Čia i ir j – mazgų numeriai, n – grafo mazgų skaičius, o matricos elementas $p_{ij}$ nusako panaudotų pralaidumo resursų kiekį tarp viršūnių i ir j.

Pageidaujamą resursų rezervą tarp dviejų mazgų pateikiame gretimumo matrica:

$$\{al_{ij} | i=\overline{1,n}\ ;\ j=\overline{1,n}\} \qquad (3)$$

Čia i ir j – mazgų numeriai, o matricos elementas $al_{ij}$ nusako užduodamą resursų rezervo kiekį tarp viršūnių i ir j.

## 2.3 Tinklo apkrautumo optimizavimo uždavinys

Tinklo matematinis modelis yra grafas. Tinklo linijų apkrautumą išreiškia grafo briaunų svoriai. Todėl tinklo apkrautumo optimizavimo uždavinys sprendžiamas randant trumpiausią maršrutą kiekvienam srautui tinklo apkrautumo grafe tarp viršūnių, nusakančių galinius srautų mazgus. Perskaičiavus visus tinklo srautus gaunamas vienodas santykinis tinklo apkrautumas, t.y. grafo briaunų, vaizduojančių tinklo linijų apkrautumą, svoriai bus vienodi. Norint maršrutizuoti naują srautą grafe tarp mazgų A ir B, ieškoma trumpiausio (optimalaus) t.y. santykinai mažiausiai apkrauto kelio tinklo apkrautumo grafe.

Sprendžiant optimizavimo uždavinį laikoma, kad pradiniu momentu esama tinklo konfigūracija aprašoma matricomis (1), (2) ir (3), pateiktomis 2.2 skyriuje. Sprendiniui gauti suformuojame tinklo apkrautumo grafo gretimumo matricą

$$\{g_{ij} | i=\overline{1,n}\,;\, j=\overline{1,n}\} \qquad (4)$$

Čia i ir j – mazgų numeriai. Matricos elementas $g_{ij}$ apskaičiuojamas iš matricų (1), (2) ir (3) pagal formulę:

$$g_{ij} = \left( \frac{p_{ij} + al_{ij}}{r_{ij}} \right), \qquad (5)$$

Suformavę grafą aprašančią matricą $\{g_{ij} | i=\overline{1,n}\,;\, j=\overline{1,n}\}$, ieškome trumpiausio kelio tarp dviejų nagrinėjamų mazgų. Rastas kelias bus laikomas optimaliu keliu tarp nagrinėjamų mazgų sukurti naujam panaudoto srauto resursui $p_{i\,j}$.

Formulės (5) fizinę prasmę galėtume apibrėžti taip: briauna tarp mazgų i ir j bus didesnio svorio, kai SDH tinkle santykinio apkrautumo ir santykinio pageidaujamo rezervo suma tarp mazgų i ir j yra didesnė. Pageidaujamo rezervo $al_{ij}$ ir jau panaudotų resursų $p_{ij}$ kiekis negali viršyti instaliuotų resursų $p_{i\,j}$ kiekio.

## 2.4 Grafų aprašymo būdai

### *2.4.1 Gretimumo matrica*

„Grafo G=(V, U) gretimumo matrica yra kvadratinė n-osios eilės matrica S=[$s_{ij}$], i=$\overline{1,n}$, j=$\overline{1,n}$, kurios elementas $s_{ij}$ apibrėžiamas taip:

$$s_{ij} = \begin{cases} 1, \text{jei virūnės i ir j yra gretimos} \\ 0, \text{priešingu atveju} \end{cases} \quad (6)$$

Neorientuotojo grafo gretimumo matrica yra simetrinė, o orientuotojo – nesimetrinė. Gretimumo matricos i-tojoje eilutėje vienetų skaičius yra lygus i-tosios viršūnės laipsniui neorientuotojo grafo atveju ir išėjimo puslaipsniui – orientuotojo grafo atveju.“ [1]

Gretimumo matricos dydis – $n^2$. Kreipties į bet kurį jos elementą laikas yra minimalus, kadangi bet kurio statinės struktūros elemento adresas yra žinomas.[4] Viršūnei gretimų viršūnių paieškos operacija yra tiesinio sudėtingumo ($\Theta(n)$). Ši operacija užima laiko tarpą priklausantį tik nuo mazgų skaičiaus n.

### *2.4.2 Incidencijų matrica*

„Grafo G=(V, U) incidencijų matrica yra stačiakampė (n x m) formato matrica A=[$a_{ij}$], i=$\overline{1,n}$, j=$\overline{1,m}$, ir jos elementas $a_{ij}$ neorientuotojo grafo atveju apibrėžiamas taip:

$$a_{ij} = \begin{cases} 1, \text{ jei i-oji virūnė incidentiška j-ajai briaunai} \\ 0, \text{ priešingu atveju} \end{cases} \quad (7)$$

Kaip ir gretimumo matricos atveju, incidencijų matrica turi n x m elementų ir yra reta“ [1].

Viršūnei gretimų viršūnių paieška yra didelio skaičiavimų sudėtingumo ($\Theta(n^2)$), tačiau šiai operacijai sugaištamas laikas priklauso nuo briaunų skaičiaus.

Pagal šiuos kriterijus incidencijų matrica yra dar blogesnis grafų vaizdavimo būdas nei gretimumo matrica ir turi daugiau teorinę nei praktinę reikšmę[1].

### *2.4.3 Briaunų matrica*

„(2 x m) formato matrica B vadinama briaunų (lankų) matrica, jei ($b_{1j}$, $b_{2j}$), j=$\overline{1,m}$ yra j-oji grafo briauna (lankas). Orientuotojo grafo atveju $b_{1j}$ žymi j-ojo lanko pradžią, o $b_{2j}$ – j-ojo lanko pabaigą.“ [1]

Saugomos informacijos kiekis – minimalus. Matricos elementų skaičius lygus 2m. Viršūnės, gretimos duotajai, paieška yra tiesinio sudėtingumo (Θ(n)) ir priklauso nuo grafo briaunų kiekio[1].

### *2.4.4 Gretimumo struktūra*

„Gretimumo struktūra – tai viršūnėms gretimų viršūnių aibių (viršūnių aplinkų) šeima.“[1] Gretimumo struktūrą patogiausia realizuoti dinaminiais sąrašais, kadangi viršūnėms gretimų viršūnių kiekis yra skirtingas.

Saugomas informacijos kiekis yra 2n, kur n – grafo viršūnių skaičius. Kreipties į bet kurią viršūnę laikas yra lygus konstantai, o viršūnei gretimos viršūnės paieška neviršija tiesinio sudėtingumo. Dėl šių priežasčių gretimumo struktūra laikoma geriausiu metodu saugoti nagrinėjamo uždavinio grafų struktūrą.

### *2.4.5 Briaunų (lankų) ir jų adresų masyvai(Tiesioginių nuorodų masyvai)*

„Briaunų (lankų) masyvas L turi 2m elementų (neorientuotiems grafams) ir m elementų (orientuotiems) grafams. Jis sudaromas taip: nuosekliai pradedant pirmąja viršūne ir baigiant paskutiniąja, iš eilės kiekvienai viršūnei surašomos jai gretimos viršūnės, t.y. pirmasis masyvas L gaunamas iš nuoseklaus peržiūrėjimo masyvo P pašalinus neigiamus elementus. Turint tik masyvą, L sužinoti viršūnes, gretimas viršūnei k yra neįmanoma. Todėl įvedamas antras – viršūnių adresų masyvas lst, turintis n+1 elementą. Šis masyvas sudaromas taip:

lst[1]:=0;

lst[i+1]:=lst[i]+d[i], $i=\overline{1,n}$,

čia d[i] – i-osios viršūnės laipsnis. Aišku, kad lst[k] parodo kiek reikia praleisti masyvo L elementų, kad rastume viršūnes, gretimas viršūnei k“[1].

## 2.5 Trumpiausių kelių grafuose paieškos algoritmai

### *2.5.1 Algoritmų sudėtingumo nustatymo metodika*

„Pagrindiniu sprendžiamo uždavinio parametru bus laikomas jo skaičiavimų sudėtingumas (arba tiesiog sudėtingumas), t.y. blogiausiu atveju algoritmo vykdomų žingsnių skaičius, išreikštas priklausomybe nuo duomenų apimties. Tarkime jei algoritmo įėjimo duomenys yra bet

kuris grafas G=<V, E>, tai užduoties apimtimi galime laikyti dydį |V|. Tada algoritmo sudėtingumu laikoma funkcija f, tokia, kad f(n) lygi maksimaliam žingsnių skaičiui apdorojant laisvai pasirinktą grafą su viršūnių kiekiu n.

Palyginkime funkcijas f(n) ir g(n), kurios aprašo 2 skirtingų algoritmų skaičiavimų apimtį apdorojant laisvai pasirinktą grafą su viršūnių kiekiu n. Lyginant funkcijų f(n) ir g(n) (su neneigiamomis reikšmėmis) kitimą labai patogios šios išraiškos:

$f(n)=\Theta(g(n)) \Leftrightarrow$ egzistuoja konstantos C, N > 0, tokios, kad $f(n) \leq C \cdot g(n)$ visiems $n \geq N$.

$f(n)=\Omega(g(n)) \Leftrightarrow$ egzistuoja konstantos C, N > 0, tokios, kad $f(n) \geq C \cdot g(n)$ visiems $n \geq N$.

Iš šių išraiškų seka, kad $f(n)=\Omega(g(n))$ tada ir tik tada, kai $g(n)=\Theta(f(n))$. Simbolių $\Theta(g(n))$ ir $\Omega(g(n))$ prasmė yra atitinkamai: eilės, nemažesnės nei g(n) ir eilės nedidesnės nei g(n). Jeigu algoritmo sudėtingumas yra $\Theta(g(n))$, tai sakome, kad šis algoritmas sprendimui sugaišta laiko proporcingai dydžiui $\Theta(g(n))$.“ [2]

### *2.5.2 Bendri paieškos grafuose principai*

„Dauguma žinomų trumpiausių kelių tarp dviejų fiksuotų viršūnių s ir t algoritmų remiasi veiksmais, kuriuos bendrais bruožais galime apibrėžti tokiu būdu: duotai lankų matricai A[u, v], $u, v \in V$, skaičiuojami apribojimai D[v] nuo s iki visų viršūnių $v \in V$. Kaskart nustačius, kad D[u]+A[u, v]<D[v], įvertį D[v] pageriname: D[v]:=D[u]+A[u, v]. Skaičiavimai nutraukiami, kai neįmanomas nei vieno iš apribojimų pagerinimas“ [2]

### *2.5.3 Fordo ir Belmano algoritmas*

Uždavinys: atstumo nuo šaltinio iki visų likusių viršūnių radimas.

Duomenys: Orientuotas n viršūnių grafas <V, E> su išskirtu šaltiniu $s \in V$, visų lankų matrica A[u, v], $u, v \in V$. Grafo briaunų ilgiai gali būti tik teigiami.

Rezultatas: Atstumas nuo šaltinio iki visų grafo viršūnių: D[v]=d(s,v), $v \in V$.

```
„1  begin
2   for v∈V do D[v]:=A[s, v]; D[s]:=0;
3   for k:=1 to n-2 do
4           for v∈V\{s} do
5                   for u∈V do D[v]:=min(D[v], D[u]+A[u,v])
6   end“[2]
```

Algoritmo skaičiavimų sudėtingumas yra $\Theta(n^3)$. [2]

### *2.5.4 Dijkstros algoritmas*

Uždavinys: atstumo nuo šaltinio iki visų likusių viršūnių radimas grafe su neneigiamo ilgio briaunomis.

Duomenys: Orientuotas n viršūnių grafas <V, E> su išskirtu šaltiniu $s \in V$, visų lankų matrica A[u, v], u, v $\in$ V. Grafo briaunų ilgiai gali būti tik neneigiami.

Rezultatas: Atstumas nuo šaltinio iki visų grafo viršūnių: D[v]=d(s,v), v $\in$ V.

„1 begin
2 for $v \in V$ do D[v]:=A[s,v]; D[s]:=0;
3 T:=V\{s}
4 while $T \neq \emptyset$ do
5 begin
6 u:=bet kuri viršūnė $r \in T$, tokia, kad D[r]=min{D[p]: $p \in T$}
7 T:=T\{u};
8 for $v \in T$ do D[v]:=min(D[v], D[v]+A[u,v])
9 end
10 end“[2]

„Tinkamai parenkant duomenų struktūras galima gauti algoritmo variantą su $\Theta(m \log n)$ sudėtingumu. Tam reikia aibę T išreikšti binariniu medžiu su aukščiu $\Theta(\log n)$ ir su savybe, kad tarp bet kurių dviejų jo viršūnių u ir v egzistuoja ryšys:

Jei u yra v sūnus, tai $D[u] \leq D[v]$ “[2] (8)

## 2.6 Išvados

- SDH tinklo apkrautumo modelis yra grafas. Patogiausia pateikimo forma – gretimumo struktūra
- Duomenų struktūra gali turėti didelę įtaką skaičiavimų apimčiai.
- SDH tinklų apkrautumo optimizavimas ekvivalentus grafų teorijos uždaviniui – rasti trumpiausią kelią nuo šaltinio iki visų likusių viršūnių. Šio kelio fizinė prasmė SDH tinkluose pateikta 2.3 skyriuje

- Išanalizavus trumpiausio kelio apskaičiavimo algoritmus, nustatyta, kad geriausiai uždavinys sprendžiamas Dijkstros algoritmu. Taikant tinkamiausias duomenų struktūras, algoritmo sudėtingumas yra $\Theta(\log n)$ arba $\Theta(n^2)$. Tačiau šis rezultatas praktikoje yra nepriimtinas, kadangi realaus tinklo mazgų skaičius gali siekti keletą tūkstančių, o esant tokiam skaičiavimų sudėtingumui, sprendimas gali būti pasiektas per laiko tarpą, netenkinantį tinklo vartotojų lūkesčių.

# 3 TRUMPIAUSIO KELIO TARP DVIEJŲ VIRŠŪNIŲ RADIMO ALGORITMO MODIFIKACIJOS

## 3.1. Algoritmo įtaka skaičiavimų sudėtingumui

Iš anksčiau pateiktų algoritmų skaičiavimų apimties požiūriu geriausias Dijkstros algoritmas. Jo sudėtingumas yra $\Theta(n^2)$. Taip pat šis algoritmas priimtiniausias, nes jis sprendžia vieno šaltinio uždavinius, t.y. vykdo optimalaus kelio paieškos nuo vieno mazgo iki visų likusių uždavinį. Mūsų atveju tai yra labiau tinkamas algoritmas, nes paieškos programa turi rasti trumpiausią kelią tarp dviejų mazgų. Kiti nagrinėti matriciniai algoritmai randa trumpiausius kelius tarp visų grafo viršūnių tarpusavyje. Tai susiję su papildomais skaičiavimais, o tai reiškia ir su didesniu sudėtingumu.

## 3.2. Duomenų struktūros įtaka skaičiavimų sudėtingumui

Tinkamos duomenų struktūros pasirinkimas gali iš esmės pakeisti skaičiavimų apimtį, nepaisant to, kad jos pakeitimai gali neturėti jokios reikšmės pačiam skaičiavimų sudėtingumui. Nagrinėdami šį klausimą įsiveskime dydį VVL – vidutinį viršūnės laipsnį. Jungiuose neturinčiuose kilpų grafuose šis dydis priklausys intervalui:

$$\mathrm{VVL(G)} \in \left[\left(2-\frac{2}{n}\right);(n-1)\right]; \qquad (4)$$

čia n – grafo viršūnių skaičius, G – grafas.

Nagrinėjamu SDH tinklo atveju grafas bus jungus ir be kilpų.

### *3.2.1 Statinė Dijkstros algoritmo realizacija*

Statiškai realizuojant algoritmą iškyla maksimalaus išsprendžiamo uždavinio apribojimas. Be specialių priemonių matricos, skirtos skaičiavimams, maksimalus dydis netinkamas realioms sistemoms. Taikant specialias atminties išskyrimo priemones, įmanoma išskirti reikiamą atminties kiekį, tačiau tai susiję su žemu sistemos resursų išnaudojamumu, kitaip tariant, su dideliu perteklinių duomenų kiekiu. Tačiau šis realizacijos būdas turi ir privalumų:

- lengva atlikti kreipties operacijas;
- itin paprasta programinė realizacija;

- patogus duomenų ir rezultatų atvaizdavimo formatas.

Statinė duomenų struktūros pavyzdys, pateiktas 7 pav., skirta Dijkstros algoritmo realizacijai gretimumo struktūros matrica (žr. 2.4.4 skyrių). Maksimalus šios struktūros talpinamas uždavinys – 1000 mazgų, kadangi kaip matome išskirta 1000 adresų viršūnių talpinimui.

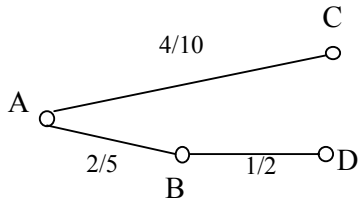
M

| | | | |
|---|---|---|---|
| M[1,1] | M[1,2] | ... | M[1,1000] |
| M[2,1] | M[2,2] | ... | M[2,1000] |
| ... | ... | ... | ... |
| M[1000,1] | M[1000,2] | ... | M[1000,1000] |

**7 pav. Statinė duomenų struktūra Dijkstros realizacijai**

### *3.2.2 Dinaminė Dijkstros algoritmo realizacija*

Dinaminės Dijkstros algoritmo realizacijos ypatumai yra priešingi statinei. Sistemos resursai yra išnaudojami optimaliai – kiekis resursų, kurį būtina išskirti norint išspręsti uždavinį, yra ribojamas tik sistemos maksimaliais pajėgumais. Tačiau dinaminė matricos realizacija neleidžia taip patogiai pasiekti bet kokį jos elementą. Jei statiniu atveju pasiekimas realizuojamas vienu veiksmu, tai dinaminiu reikalinga veiksmų seka. Tai be abejo įneša laiko nuostolių, tačiau sutaupo sisteminių resursų. Paveikslėlyje Nr. 9 parodytas pavyzdys, kaip mažo VVL grafuose dinaminė struktūra gali sutaupyti daug sisteminių resursų.

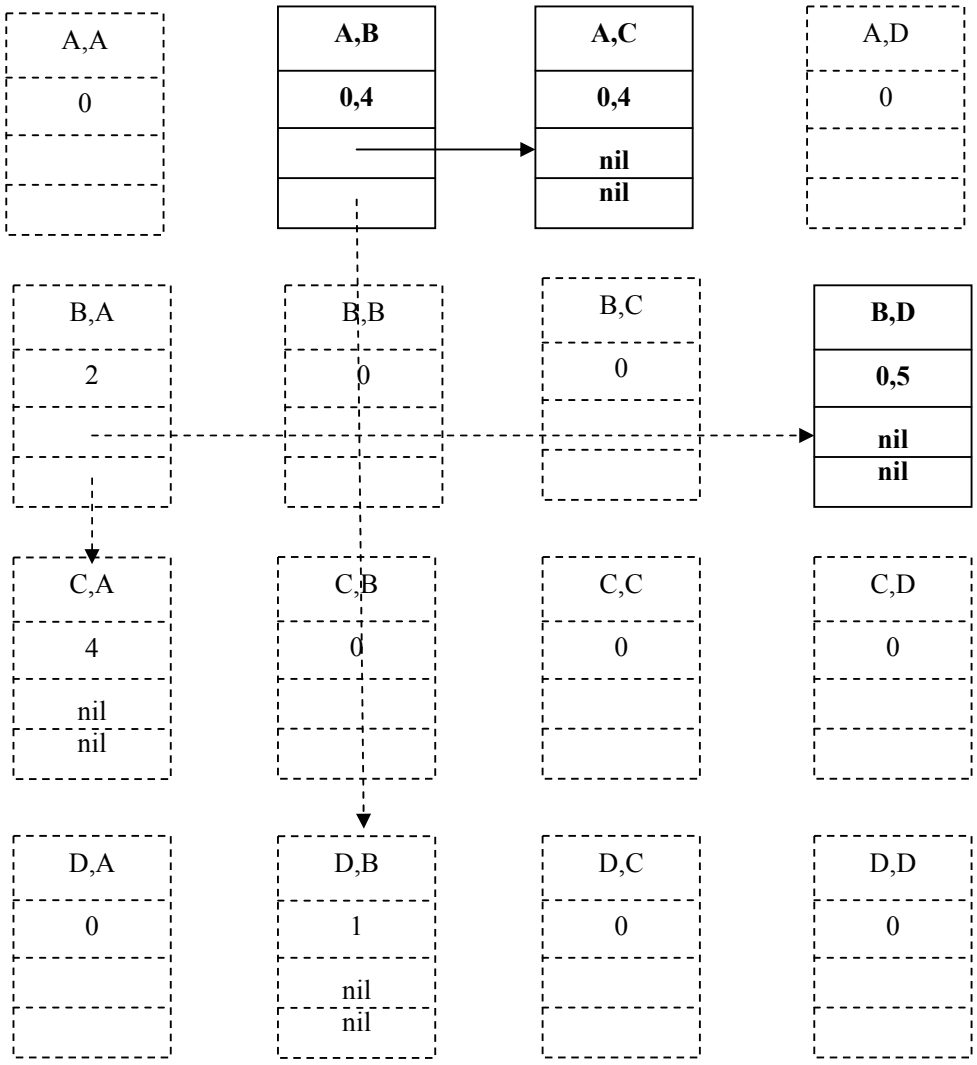



**8 pav. Minimalaus VVL svorinio grafo pavyzdys**

Šio grafo briaunų svoriai nustatomi analogiškai kaip pateikta skyriuje 2.1.

**3 lentelė gretimumo struktūra 8 pav. pateiktam grafui**

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 0,4 | 0,4 | ∞ |
| B | 0,4 | 0 | ∞ | 0,5 |
| C | 0,4 | ∞ | 0 | ∞ |
| D | ∞ | 0,5 | ∞ | 0 |




**9 pav. Dinaminės gretimumo struktūros pavyzdys Dijkstros realizacijai**

Anksčiau pavaizduotoje duomenų struktūroje kiekvienas elementas (pavaizduotas stačiakampiu) programine prasme yra objektas, saugantis gretimas viršūnes nagrinėjamoms viršūnėms. Taip pat kiekvienas objektas saugo briaunos, jungiančios nagrinėjamas viršūnes,

svorį. Gretimų objektų pasiekiamumui yra naudojamos nuorodos, pavaizduotos punktyrinėmis ir vientisomis rodyklėmis.

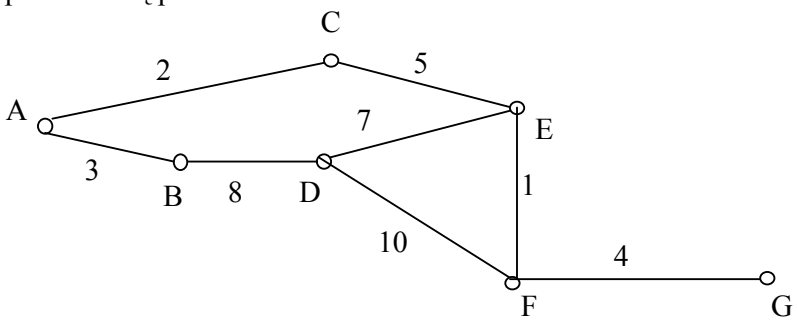
Punktyrais pavaizduoti objektai yra pertekliniai ir faktiškai atmintyje nesukuriami. Žemiau pagrindinės įstrižainės objektai būtų reikalingi, jei grafas būtų orientuotas. Bendru atveju SDH tinklo srautai gali būti ir vienkrypčiai ir dėl to juos atvaizduojantys grafai turėtų būti orientuoti. Tačiau nagrinėjant užduotį nutarta, kad galima įsivesti vidinį grafo neorientuotumo apribojimą.

Vykdant Dijkstros algoritmą dinaminė gretimumo struktūros matrica palaipsniui didėja ir, jei grafas jungus, baigus algoritmą turėtų būti sukurti visi elementai esantys aukščiau pagrindinės įstrižainės. Algoritmo vykdymo pradžioje dinaminė struktūra sutaupytų daug laiko, kadangi trumpiausio rasto kelio paieška artėtų prie konstantos, o bendras algoritmo vykdymo sudėtingumas prie tiesinio. Tačiau vėliau, didėjant dinaminei matricai, skaičiavimų apimties požiūriu vis sudėtingėja kreipties operacijos ir bendras laikas tampa blogesnis nei statiniu atveju. Kreipties laikui artėjant prie tiesinio bendras algoritmo sudėtingumas artėja prie $\Theta(n^2)$.

### *3.2.3 Mišri Dijkstros algoritmo realizacija*

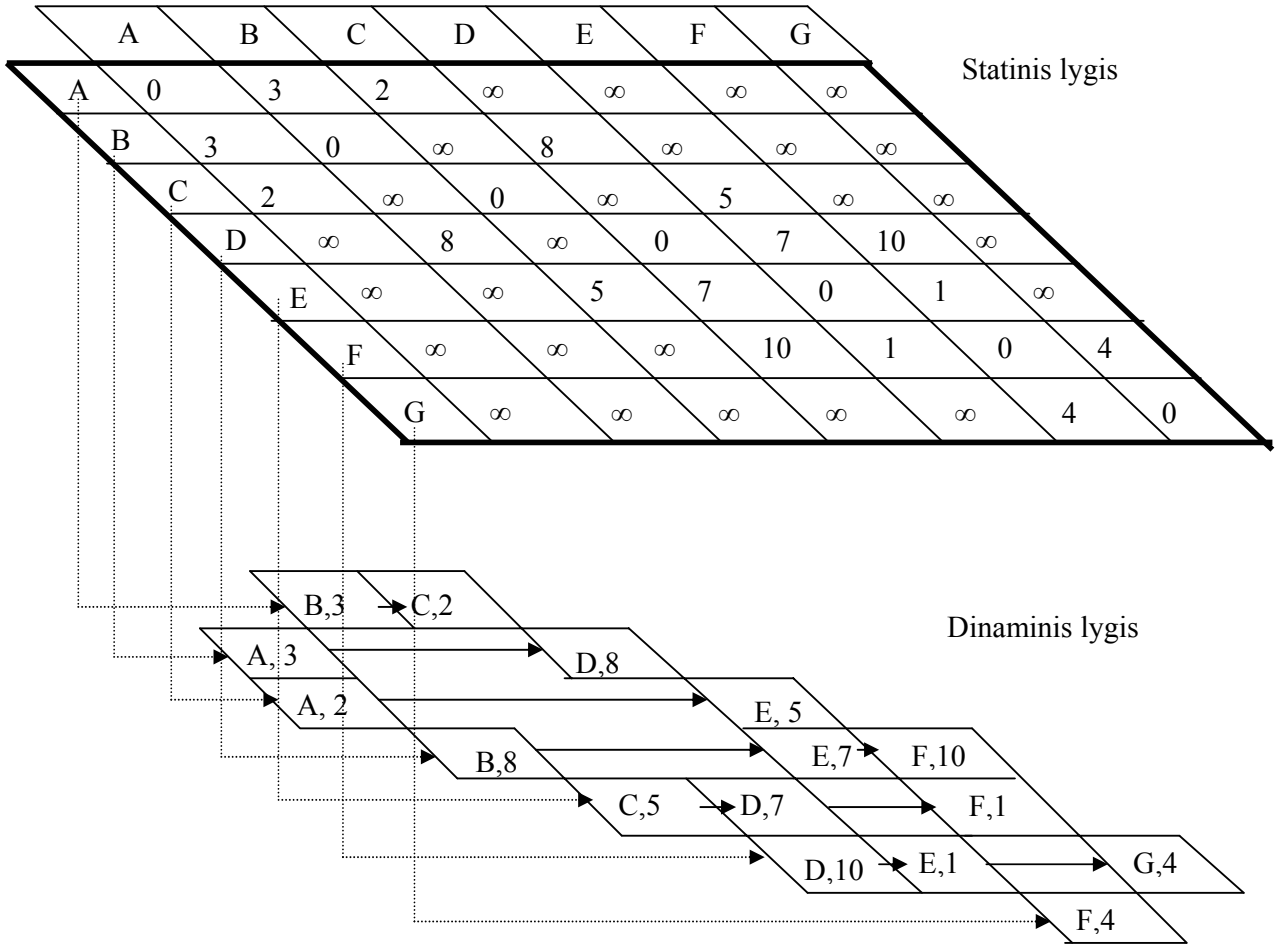
Mišrios Dijkstros algoritmo realizacijos esmė – statinių duomenų struktūrų naudojimas kreipties operacijoms ir dinaminių duomenų struktūrų naudojimas paieškos operacijoms. Ši metodika sujungia abiejų anksčiau minėtų algoritmų teigiamas vykdymo greičio savybes. Tačiau šis algoritmas reikalauja tiek atminties resursų kiek abu anksčiau minėti metodai kartu sudėjus. Taip pat mišri realizacija iš statinės realizacijos „paveldi“ maksimalios leistinos išskirti statinės atminties kiekį. Mišrios realizacijos skaičiavimų sudėtingumas įvairiose algoritmo vykdymo stadijose yra skirtingas. Paieškos trukmė kinta nuo nykstamai mažo iki tiesinio sudėtingumo. Kreipties trukmė, skirtingai nei dinaminiu atveju, yra visada konstanta. Sudėjus bendrą algoritmo vykdymo laiką, minimalus sudėtingumas artėja prie tiesinio $\Theta(n)$, o maksimalus neviršija kvadratinio $\Theta(n^2)$.

Tarkime turime grafą, pavaizduotą pav. 10.




**10 pav. SDH struktūrą vaizduojančio grafo pavyzdys**

Šiam grafui suformuojame mišrią duomenų struktūrą:




**11 pav. Mišrios duomenų struktūros pavyzdys**

Kaip matome 11pav. statinį mišrios struktūros lygį sudaro grafų vaizdavimo gretimumo struktūra realizuota statiškai, o dinaminį lygį – tokia pati duomenų struktūra tik realizuota dinamiškai. Dinaminiame lygyje egzistuoja tik tie elementai, kurie sieja gretimas viršūnes. Dinaminį ir statinį lygį sieja nuorodos: į kiekvieną dinaminio lygio eilutę, vaizduojančią gretimas viršūnes nagrinėjamai viršūnei, ateina nuoroda iš statinio lygio viršūnės, kuriai minėtos viršūnės yra gretimos.

Tokia duomenų struktūra yra pasiekiamas maksimalus kreipties į bet kurią viršūnę ir bet kurios viršūnės paieškos laikas. Jei yra reikalinga kreiptis į bet kurį elementą, tai naudojamės statiniu lygiu. Šis veiksmas užtrunka $\Theta(0)$. Jei yra reikalinga bet kurios viršūnės paieška, tai per nuorodą tarp lygių pereiname į dinaminį lygį ir einame per dinaminį sąrašą, kol randame

ieškomą viršūnę. Šis veiksmui atlikti vidutiniškai turėtume peržiūrėti $\frac{n}{2}$ elementų, tačiau bendrąja prasme sudėtingumas būtų Θ(n).

Esant nedideliam grafų VVL, dinaminis lygis, aprašantis pradinę grafo struktūrą, yra santykinai mažas. Jį (10 pav.) sudaro vos keletas komponentų. Tačiau šia struktūrą naudojant Dijkstros algoritmo realizacijai, vykdymo metu ši struktūra ženkliai plečiasi. Jei grafas jungus, tai gavus sprendinį dinaminio lygio užimamas atminties kiekis bus proporcingas:

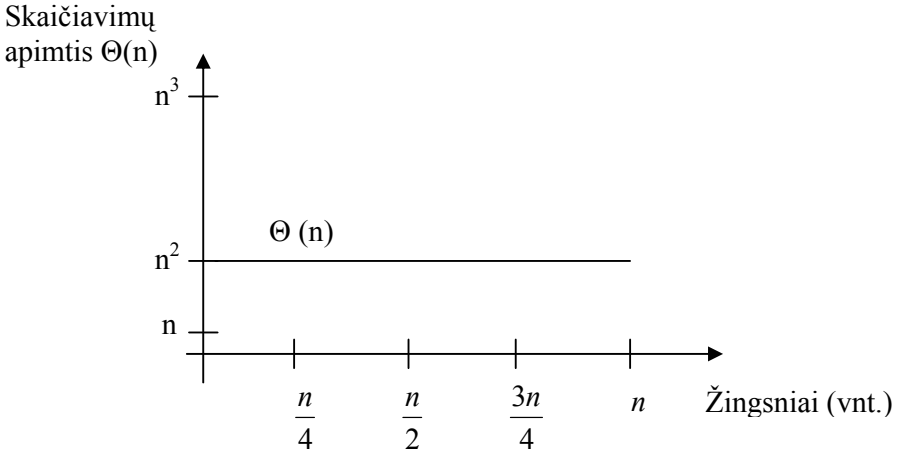
$$M_D(G) = n^2 - n \qquad (5)$$

Čia $M_D$ – užimamas atminties kiekis dinaminiame lygyje, G – nagrinėjamas grafas, n – mazgų kiekis.

Struktūros plėtimosi greitis nėra toks aktualus užimamos atminties požiūriu, kadangi mes bet kuriuo atveju žinome kokio atminties kiekio mums reikės uždavinį išsprendus. Šiuo atveju struktūros dydis bet kuriuo laiko momentu lemia uždavinio sprendimo greitį. Taip yra todėl, kad Dijkstros algoritmo dalis yra visų viršūnių perrinkimas. Jei struktūra fiksuoja mažesnį viršūnių skaičių, tai natūralu, kad apeinamų viršūnių skaičius bus mažesnis ir tuo pačiu vykdymo greitis bus didesnis.

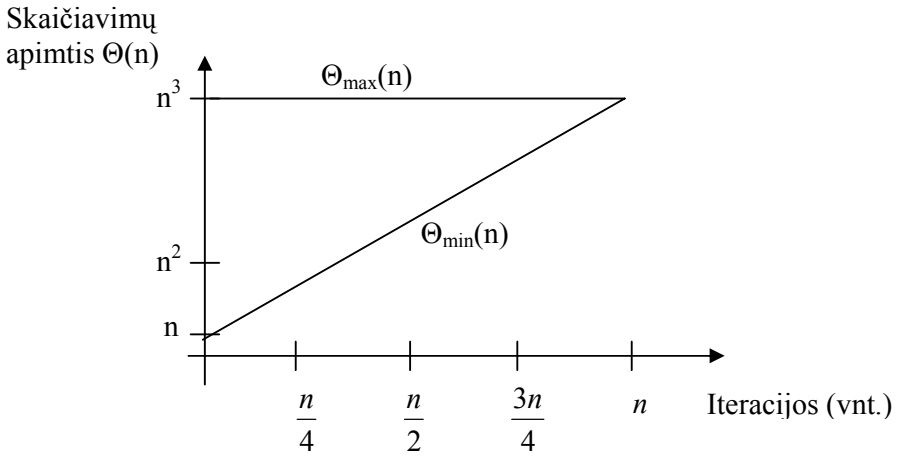
### *3.2.4 Skirtingų duomenų struktūrų įtakos vykdymo laikui palyginimas*

Realizuojant algoritmą statiškai skaičiavimų sudėtingumas, artėjant prie tikslo, nesikečia, kadangi kiekviename žingsnyje vykdomų veiksmų skaičius statinėje gretimumo struktūroje priklauso tik nuo viršūnių skaičiaus. (pav. 12)



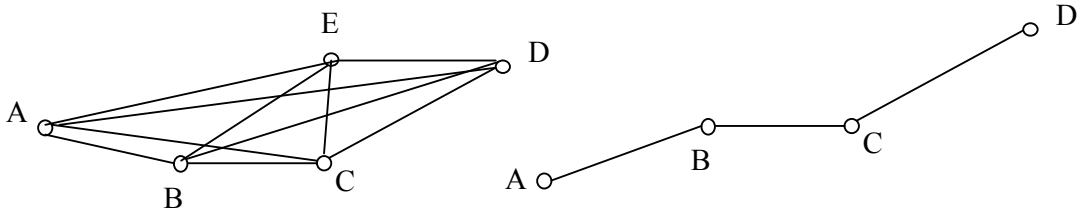

**12 pav. Statinės Dijkstros algoritmo realizacijos skaičiavimų sudėtingumo charakteristikos**

Realizuojant algoritmą dinaminiu būdu vieno žingsnio vykdymo laikas gali ženkliai skirtis. Šiuos skirtumus galime pailiustruoti grafiškai pav. 13.



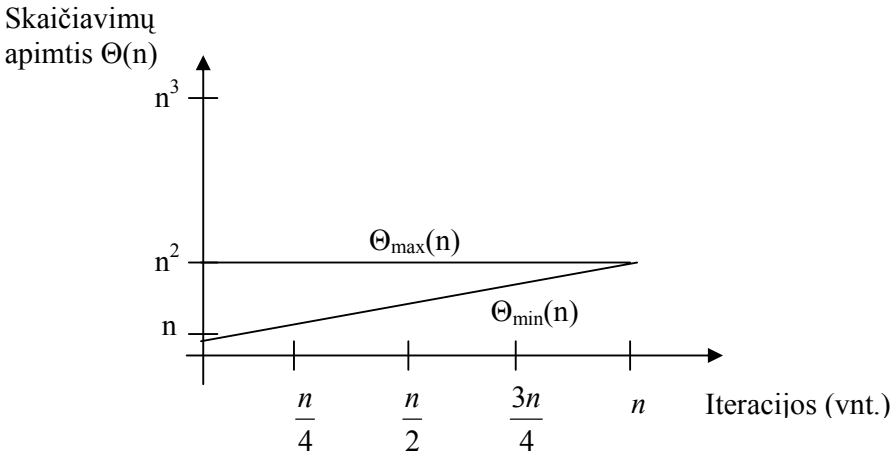

**13 pav. Dinaminės Dijkstros algoritmo realizacijos skaičiavimų sudėtingumo charakteristikos**

Kaip matome iš 13 pav. dinaminės Dijkstros algoritmo realizacijos galimi sudėtingumo skirtumai tarp geriausio ir blogiausio sudėtingumo yra labai dideli. Blogiausiu atveju algoritmas nuo vykdymo pradžios dirbs $\Theta(n^3)$ sudėtingumu. Geriausiu atveju sudėtingumas auga palaipsniui didėjant dinaminės struktūros elementų kiekiui. Šių skirtumų prigimtis – grafų VVL skirtumai. Jei mazgas, iš kurio bandoma rasti optimalius maršrutus yra gretimas visiems likusiems grafo mazgams, tai dinaminė realizacija iš karto sugeneruos maksimalią įmanomą struktūrą ir vykdys n žingsnių sudėtingumu $\Theta(n^3)$. Jei grafas yra grandinės pavidalo (bet kuris mazgas turi ne daugiau kaip 2 kaimynus), tai pradinės dinaminės struktūros dydis bus minimalus ir sieks (n-1). Ieškant trumpiausio kelio bus aptinkami vis nauji maršrutai ir dėl to sudėtingumas palaipsniui augs. Aptarti ekstremalių savybių šiam metodui grafai pavaizduoti 14 pav.




**14 pav. Grafai dinaminiu metodu sprendžiami maksimaliu ir minimaliu sudėtingumu**

Realizuojant algoritmą mišriu būdu vieno žingsnio vykdymo laikas gali šiek tiek skirtis. Maksimalus vykdymo sudėtingumas taip pat nėra didelis (pav. 15)




**15 pav. Mišrios Dijkstros algoritmo realizacijos skaičiavimų sudėtingumo charakteristikos**

Kaip matome šiuo metodu sprendžiamų uždavinių sudėtingumas bendru atveju gali siekti $\Theta(n^2)$, tačiau pradinėse stadijose, esant tinkamai grafo struktūrai, sudėtingumas gali būti artimas tiesiniam. Teoriškai neįmanoma grafo struktūros ypatumų apibrėžti tikimybiniu skirstiniu, kuriuo remdamiesi galėtume nusakyti kokie grafai SDH tinkluose naudojami dažniau ir kokie rečiau. Tačiau SDH tinklų specialistų teigimu praktinį SDH tinklą vaizduojančio grafo VVL yra santykinai nedidelis (dideliuose tinkluose iki 0,1n ). Taip yra todėl, kad SDH tinkle kiekvienai naujai linijai nutiesti reikalingi dideli finansiniai ištekliai. Projektuojant tinklą yra sukuriama galimybė turėti pagrindinę ir rezervinę liniją, kuria būtų nukreipiami srautai esant gedimui. Didelis tinklo šakotumas operatoriui taip pat sukelia papildomų priežiūros išlaidų, sudėtingumo peržiūrint sugeneruotus maršrutus.

## 3.3. Trumpiausio kelio paieškos algoritmas nedidelio VVL grafams

Svarbiausia užduotis kuriant alternatyvius algoritmus trumpiausio kelio paieškai yra sumažinti uždavinio sprendimo sudėtingumą iki $\Theta(n \log_2(n))$ arba net $\Theta(n)=n$. Siekiama išnagrinėti ar įmanoma sukurti metodą, tenkinantį tokius sudėtingumo reikalavimus, ir stebėti kaip kinta vartojamos atminties kiekis. Taip pat svarbu nustatyti ar galima rasti mažesnio sudėtingumo euristinį trumpiausio kelio paieškos algoritmą. Mažesnio sudėtingumo algoritmas yra labai aktualus, nes tinklo resursų perskirstymo metu jokie nauji srautai negali būti įterpiami ar pašalinami iš tinklo. Natūralu, kad užsakovo pageidavimu šis laikotarpis turi būti kaip įmanoma mažesnis.

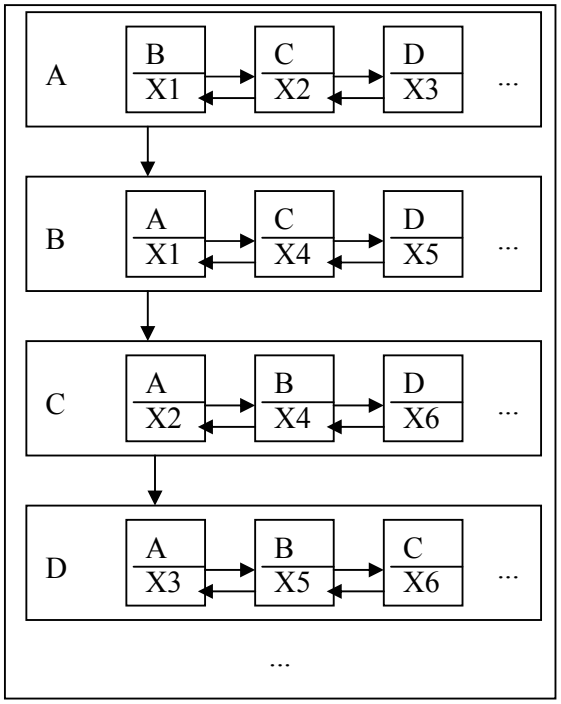
### *3.3.1 Pastovaus skaičiavimų sudėtingumo metodas*

Analizuojant Dijkstros algoritmą pastebėti tokie trūkumai:

- Atliekant kiekvieną žingsnį būtina surasti patį trumpiausią dar nenagrinėtą tarpinį kelią.
- Kiekviename žingsnyje rastas kelias turi būti palygintas su ankstesniais rastais keliais ir radus trumpesnį kelią visas maršrutas turi būti perrašytas

Išanalizavus šiuos trūkumus buvo prieita prie išvados, kad mažo VVL grafams turėtų būti geresnių būdų rasti trumpiausią kelią.

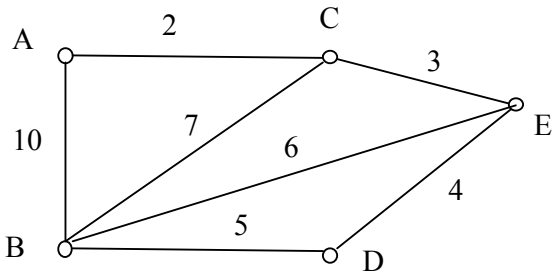
Tarkime turime G=<V, U>, kur V-grafo viršūnių sąrašas, U-grafo briaunų sąrašas. Gretimumo struktūrą aprašome dinamiškai. Pradiniai duomenys įgauna tokią formą:




**16 pav. Pradinė duomenų struktūra**

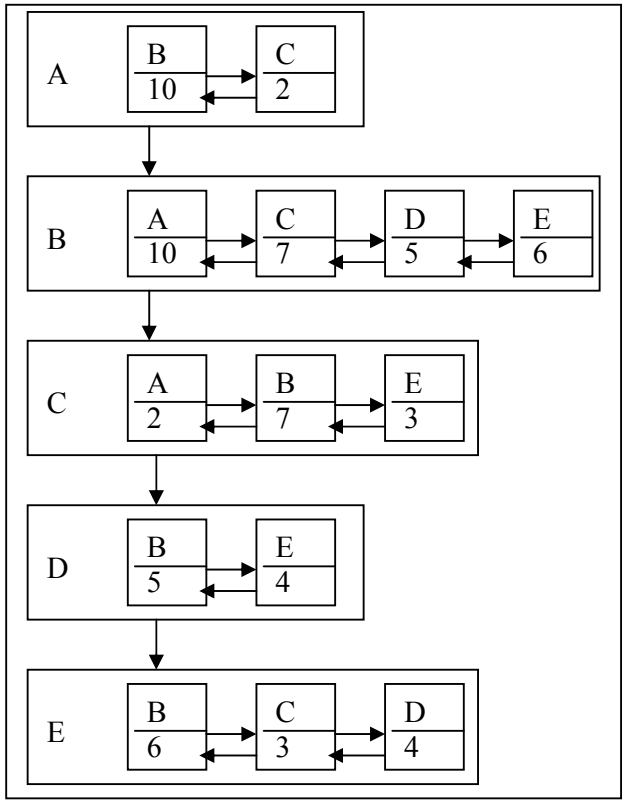
Čia A, B, C, D – viršūnės, X1, X2, X3, X4, X5, X6 – atstumai tarp atitinkamų viršūnių.

Panagrinėkime algoritmą pavyzdžiu. Tarkime turime grafą:




**17 pav. Grafo pavyzdys**

Pradinė gretimumo struktūra:



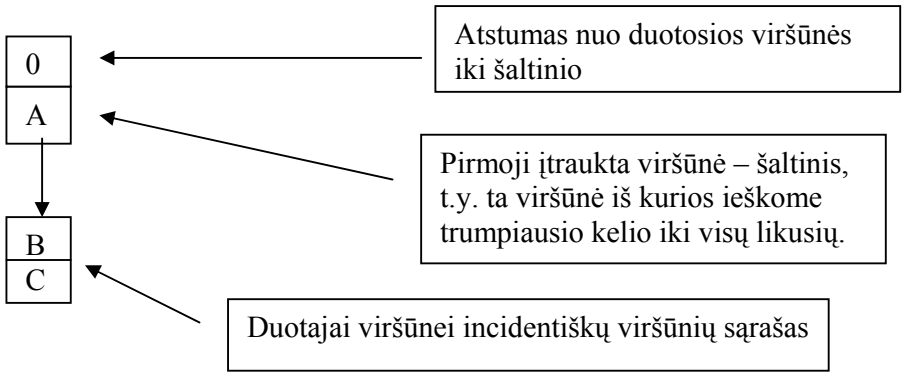

**18 pav. Pradinė grafo gretimumo struktūra**

Algoritmo darbas pradedamas nuo šaltinio. Kiekviename algoritmo žingsnyje yra paimama nauja viršūnė, gretima nagrinėjamoms ir apskaičiuojamas trumpiausias kelias iki šios viršūnės nuo jos iki šaltinio. Tame pačiame žingsnyje apskaičiuojama ar kelias per šią viršūnę iki bet kurios jai gretimos yra trumpesnis nei iki tol rastas. Viršūnė toliau nebenagrinėjama, kai peržiūrimos visos jai gretimos viršūnės.

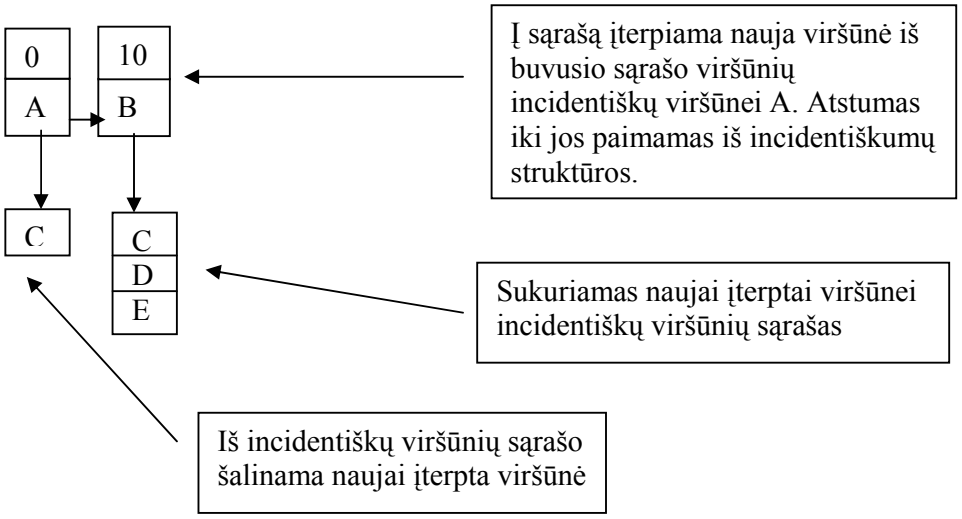
Duomenys kinta tokiu principu: pradžioje trumpiausių atstumų sąrašą sudaro tik šaltinis. Kiekviename žingsnyje pridedame po pirmą pasitaikiusią viršūnę iš gretimų anksčiau įtrauktosioms ir dar nesančią turime sąraše. Pridėję naują viršūnę, perskaičiuojame atstumą nuo

anksčiau nagrinėtų iki jos ir atvirkščiai – nuo jos iki anksčiau nagrinėtų. Įterpę naują viršūnę, susijusią su anksčiau aplankytąja briauna, šaliname šį gretimumą fiksuojančią briauną iš sąrašo.

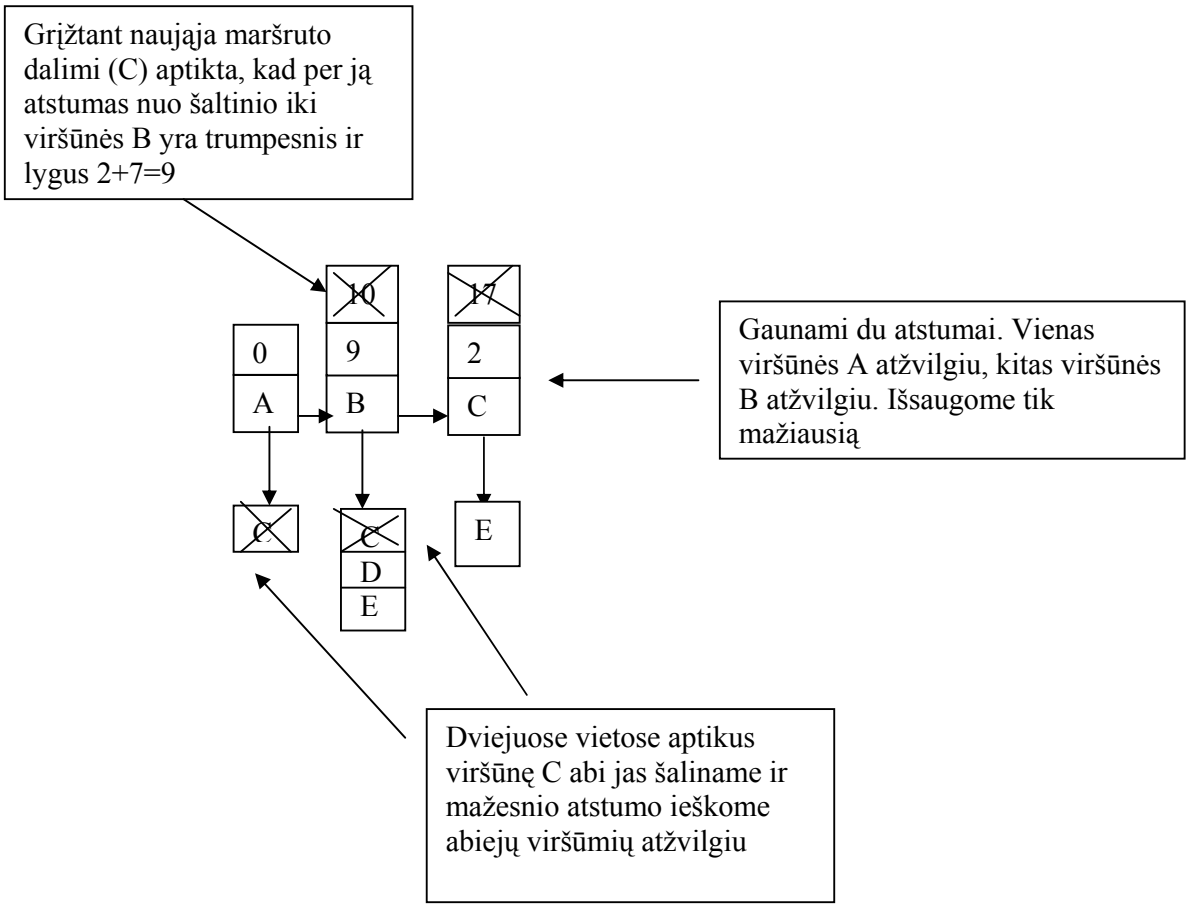
Toliau pateiksime detalų algoritmo aprašymą vykdydami jį pažingsniui:



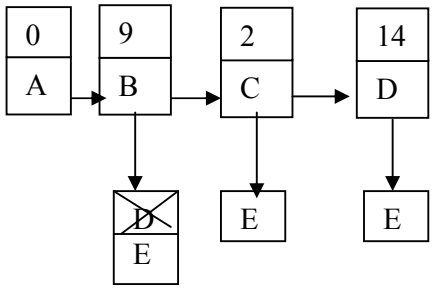

**19 pav. Pirmasis žingsnis**



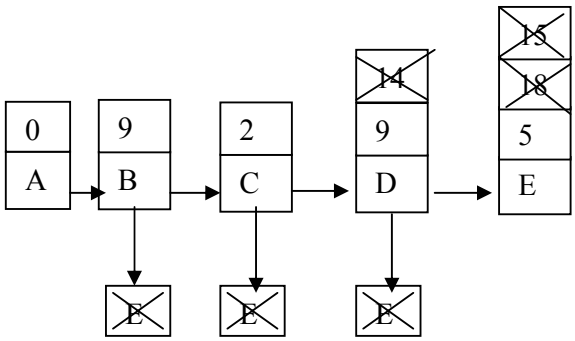

**20 pav. Antrasis žingsnis**




**21 pav. Trečiasis žingsnis**




**22 pav. Ketvirtasis žingsnis**

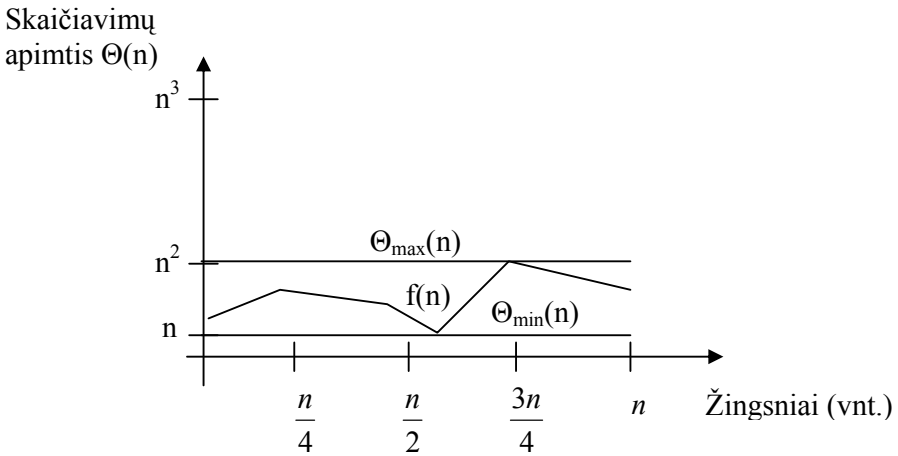



**23 pav. Penktasis žingsnis**

Kaip matome, visi veiksmai su dinaminiu sąrašu yra atliekami tik su nagrinėjamai viršūnei gretimomis viršūnėmis. Tai yra didelis privalumas mažo VVL grafuose. Dijkstros algoritmas sukurtų menamo gretimumo struktūras, kurias nuolat optimizuotų. Menamas gretimumas čia turėtų būti suprantamas kaip gretimumas per kitas viršūnes.

Kaip ir buvo siekiama algoritmo sudėtingumas vykdymo metu nebūtinai didėja. Išrinktajai viršūnei, kuri turi būti patalpinta į sąrašą gretimų viršūnių skaičius visada yra daug mažesnis nei bendras viršūnių skaičius, todėl vieno žingsnio vykdymo sudėtingumas yra tarp konstantinio ir tiesinio.

Vykdant algoritmą sudėtingumas gali ir mažėti. Tai priklauso tik nuo nagrinėjamai viršūnei gretimų viršūnių skaičiaus. Bendru atveju šis skaičius – atsitiktinis dydis intervale [1, n-1]. Todėl sudėtingumas bet kuriame žingsnyje yra atsitiktinis. Praktinis bet kurio grafo trumpiausio kelio paieškos algoritmo sudėtingumo kitimas grafiškai galėtų būti pavaizduotas kreive f(n): $\Theta_{min}(n) \le f(n) \le \Theta_{max}(n)$ intervale [0, n].




**24 pav. Skaičiavimo sudėtingumų intervalai**

### *3.3.2 Algoritmo integravimas į tiriamąjį uždavinį*

Kaip žinome Dijkstros algoritmas atlieka trumpiausių kelių iš vienos viršūnės iki visų likusių viršūnių paiešką. Pagal užduotą kelio ilgį (4 formulė) atlikus algoritmą mes gautume 1 srauto optimalų maršrutą. Tačiau, nagrinėjant aukštesnio hierarchinio vieneto srautą, reikėtų atkreipti dėmesį, kad ne visada geriausia maršrutizuoti jį visą vienu maršrutu. Kadangi SDH tinklai užtikrina srautų skaidymą ir sujungimą, tai maršrutizuojant stambesnius srautus geriausias variantas būtų juos suskaidyti į smulkiausius srautų vienetus TU-11 ir maršrutizuoti atskirai. Dalis gautų smulkiausių srautų eis vienu maršrutu. Tokius srautus galėtume sujungti ir atliekant sisteminį maršrutizavimą tuo išvengtume painiavos atsiradusios padidėjus srautų skaičiui. Taigi, jei reiktų optimaliai maršrutizuoti vieną STM-1 srautą, tai uždavinį reikėtų spręsti 84 kartus, nes vieną STM-1 srautą sudaro aštuoniasdešimt keturi TU-11 srautai.

## 3.4 Išvados

- Ištyrus Dijkstros algoritmui taikomas duomenų struktūras, nustatyta, kad uždavinio pradžioje mažiausias veiksmų skaičius yra artimas $\Theta(n)$, o uždavinio pabaigoje – $\Theta(n^2)$.
- Tinkamiausias duomenų modelis Dijkstros algoritmo realizacijai yra mišri duomenų struktūra. Blogiausiu atveju jos skaičiavimų sudėtingumas neviršija tradicinėmis duomenų struktūromis pasiekiamo dydžio $\Theta(n^2)$, o geriausiu atveju, atliekant uždavinio žingsnius, gali tiesiškai kisti nuo $\Theta(n)$ iki $\Theta(n^2)$.

# 4 PATOBULINTO ALGORITMO EKSPERIMENTINIS TYRIMAS

## 4.1 Matavimo metodika

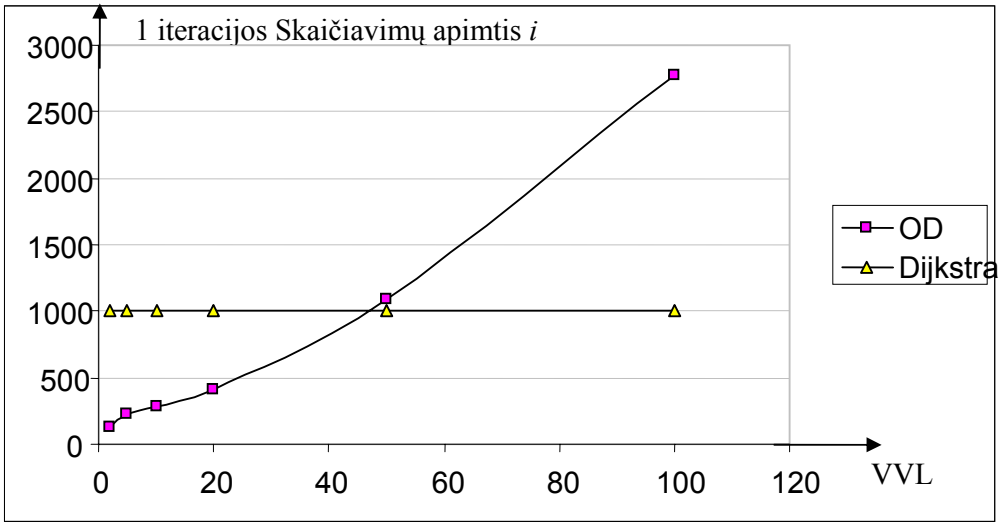
Įvertinant praktinį algoritmo efektyvumą buvo vengiama laikinių išraiškų, nes skirtingos operacijos skirtingose platformose užima skirtingą laiko tarpą. Pvz. Dinaminis atminties išskyrimas lyginant su statiniu realizuotais Java ir Delphi gali ženkliai skirtis. Skaičiavimų laikas taip pat labai smarkiai priklauso nuo realizacijos, todėl visi skaičiavimai buvo atliekami būtent skaičiavimų apimties požiūriu ir nebuvo stengtasi įvertinti laiko.

## 4.2 Eksperimento sąlygos

Eksperimento metu generuotas atsitiktinis nuo 500 iki 5000 mazgų dydžio grafas, kurio VVL yra intervale nuo 2 iki 200. Grafo briaunų svoriai yra atsitiktiniai ir tolygiai pasiskirstę intervale nuo 1 iki 5000. Trumpiausio kelio ieškoma nuo pirmosios viršūnės iki visų likusių. Eksperimentui naudojama imtis: 5 sprendiniai su skirtingais grafais. Geriausias ir blogiausias iš jų atmetami, o iš likusių 3 išvedamas aritmetinis vidurkis. Tarkime 5 eksperimentų metu gavome šiuos rezultatus: 50, 48, 63, 50, 55. Atmetame geriausią rezultatą 48 ir blogiausią rezultatą 63. Likusių 3 rezultatų aritmetinis vidurkis yra 51,67. Šis skaičius laikomas galutiniu eksperimento rezultatu. Eksperimentas vykdomas su tuo pačiu grafu naudojant abu algoritmus. Eksperimentas kartojamas visiems nagrinėjamiems VVL ir mazgų skaičiui n.

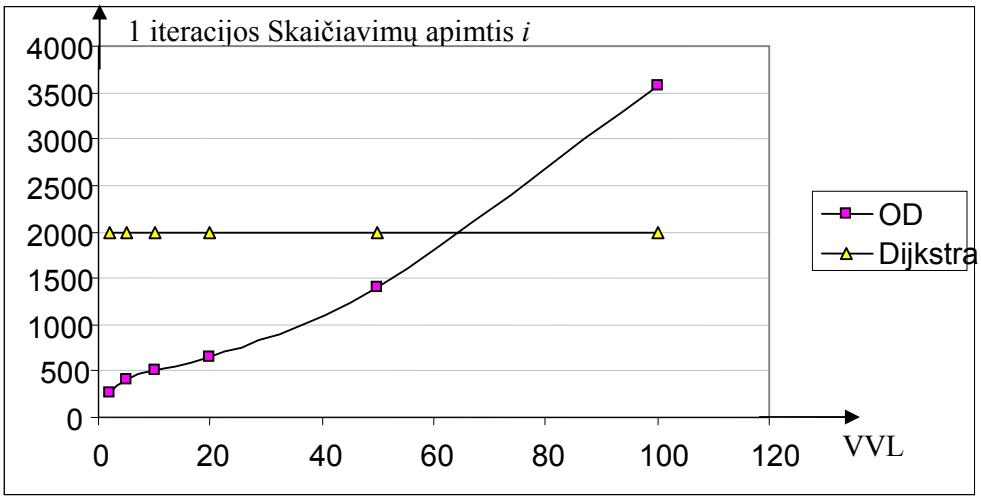
## 4.3 Skaičiavimo sudėtingumo priklausomybių grafikai

Žemiau pavaizduotose priklausomybėse pažymimos vidutinės skaičiavimų apimtys vienam žingsniui. Dijkstros algoritmo atveju tai vaizduoja pastovaus, du kartus didesnio už viršūnių skaičių, lygio tiesė. Taip yra todėl, kad viename žingsnyje vieną kartą visos viršūnės perrenkamos ieškant trumpiausio iki to žingsnio rasto kelio. Antrą kartą perrenkamos ieškant dar trumpesnio kelio per naują viršūnę. Optimizuoto algoritmo vidutinis vieno žingsnio skaičiavimų kiekis yra žymiai sudėtingesnis. Jį apima ir nagrinėjamų viršūnių skaičius ir geriausio iš rastų kelių radimas, naujų narių sukūrimas.



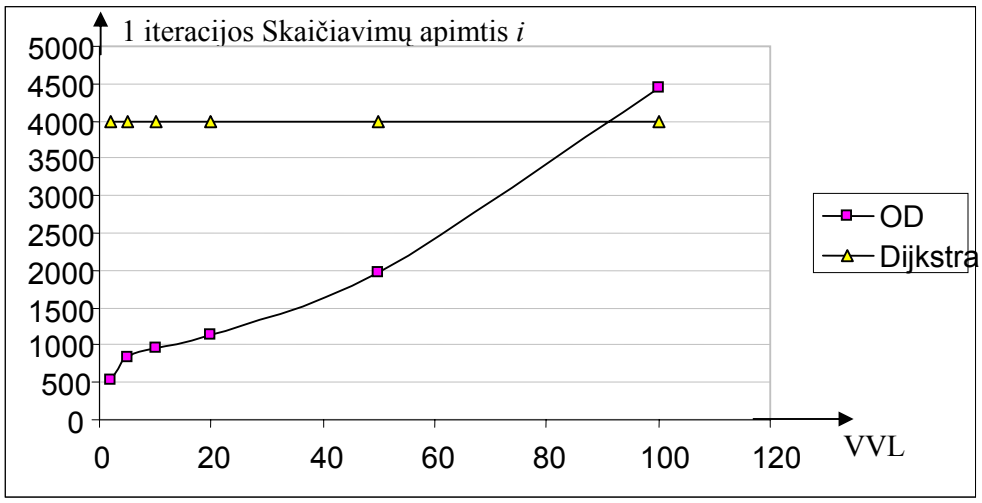

**25 pav. Skaičiavimų kiekio priklausomybė nuo VVL, kai n=500**

Kaip matome pav. 25 prie mazgų skaičiaus n=500 esant minimaliam VVL algoritmas sprendžia uždavinį 4-5 kartus greičiau nei Dijkstros algoritmas. VVL pasiekus 44 (toliau šį tašką vadinsime lūžio tašku $VVL_k$) patobulinto algoritmo skaičiavimų apimtis pasiekia Dijkstros algoritmo lygį. Toliau, didinant VVL, algoritmo greitis sparčiai krenta.



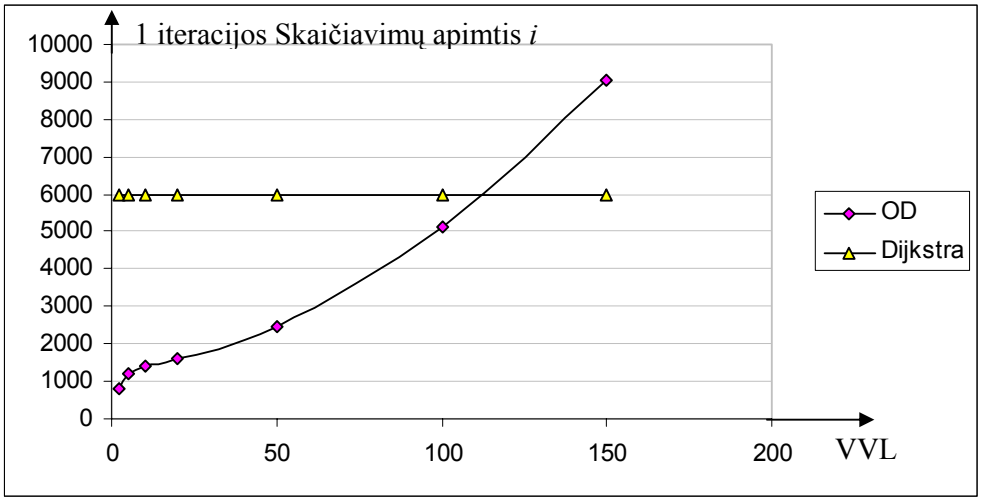

**26 pav. Skaičiavimų kiekio priklausomybė nuo VVL, kai n=1000**

Kai n=1000 (26 pav.), lūžio taškas $VVL_k$=78.



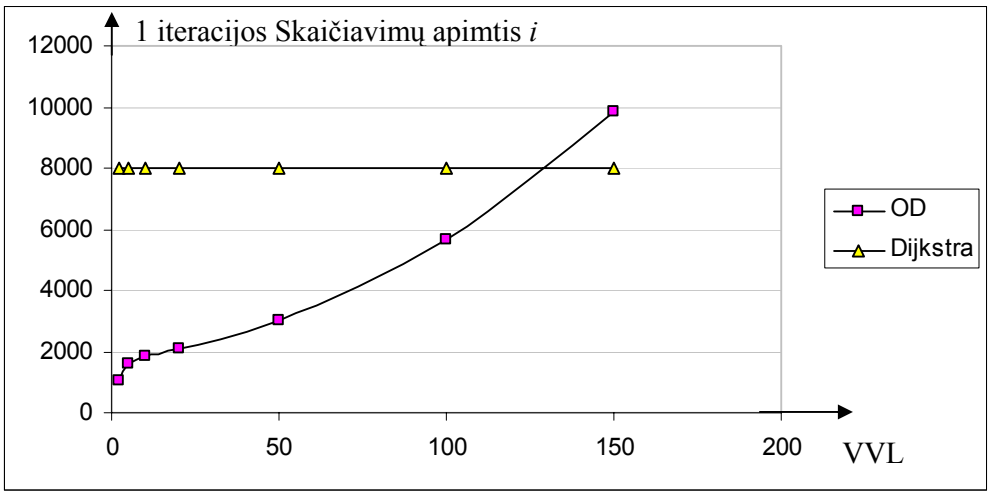

**27 pav. Skaičiavimų kiekio priklausomybė nuo VVL, kai n=2000**

Kai n=2000 (27 pav.), lūžio taškas $VVL_k$=92.



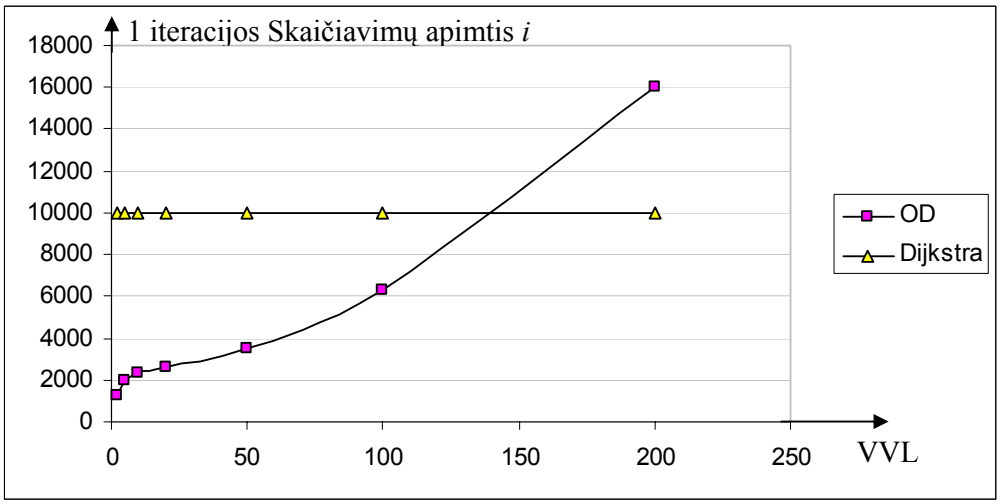

**28 pav. Skaičiavimų kiekio priklausomybė nuo VVL, kai n=3000**

Kai n=3000 (28 pav.), lūžio taškas $VVL_k$=111.




**29 pav. Skaičiavimų kiekio priklausomybė nuo VVL, kai n=4000**

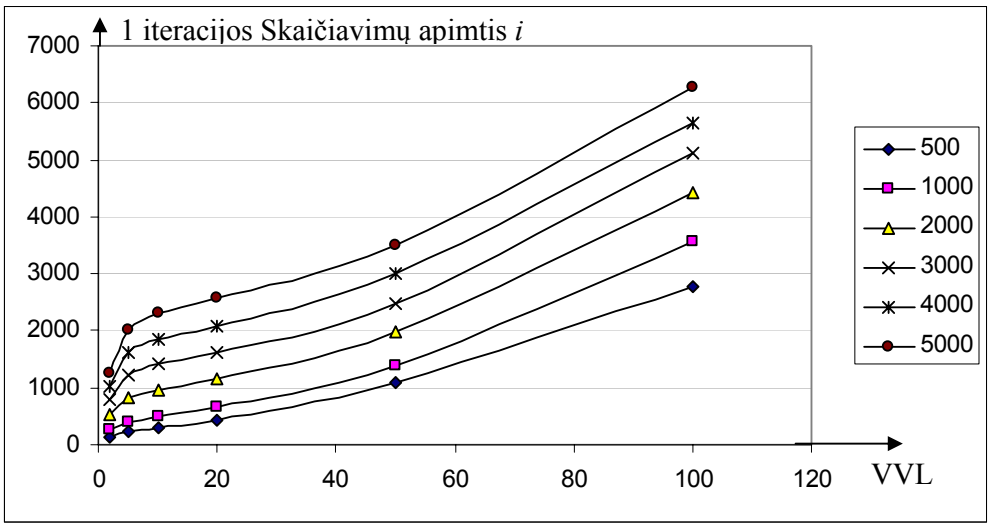
Kai n=4000 (29 pav.), lūžio taškas $VVL_k$=128.




**30 pav. Skaičiavimų kiekio priklausomybė nuo VVL, kai n=5000**

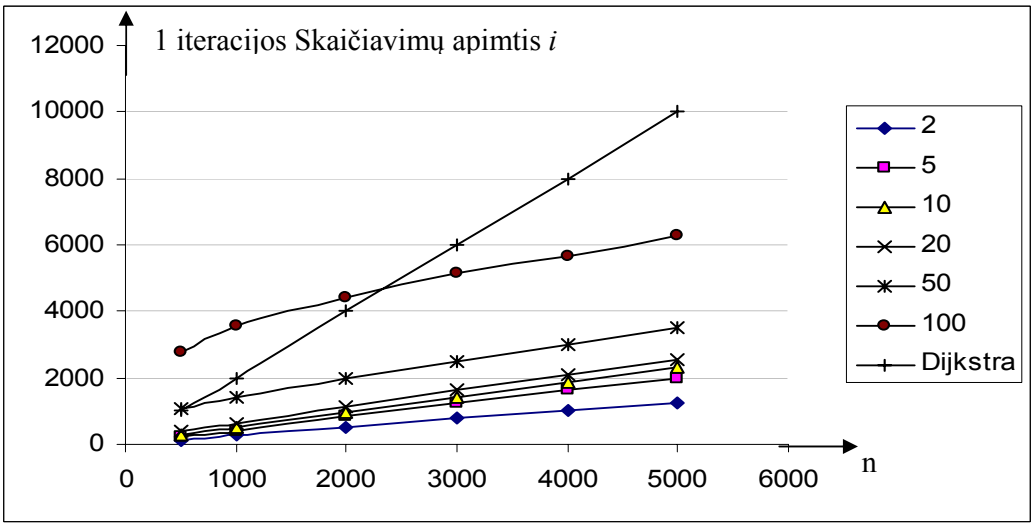
Kai n=5000 (30 pav.), lūžio taškas $VVL_k$=139.

Kaip matome, iš visų nagrinėtų eksperimentų, VVL neviršijant 15 prie bet kurio viršūnių kiekio n, patobulintas algoritmas uždavinį sprendžia 4-8 kartus greičiau nei Dijkstros algoritmas. Viršijus $VVL_k$ tašką patobulinto algoritmo greitis sparčiai prastėja.

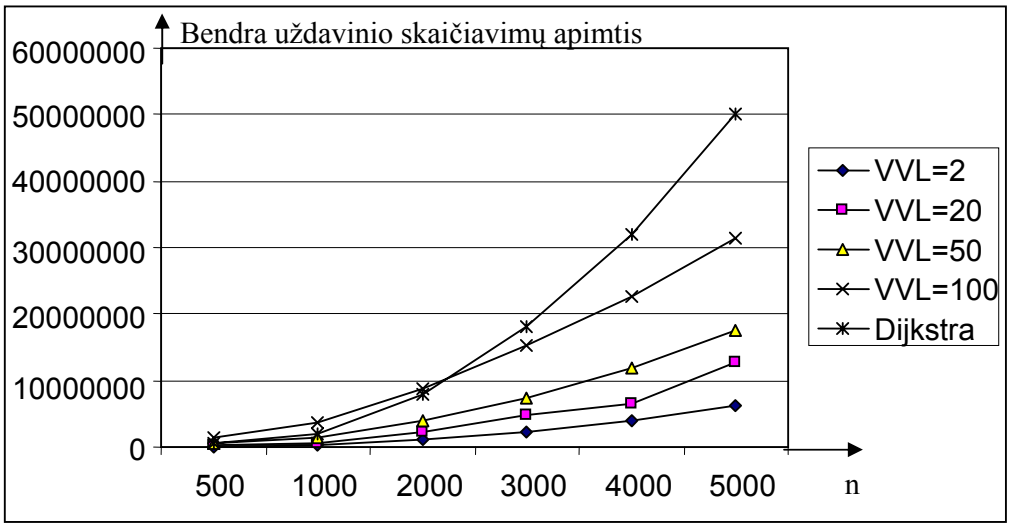



**31 pav. Skaičiavimų kiekio priklausomybė nuo VVL, kai n įvairus**

Šiame grafike (pav. 32) pavaizduotas vieno žingsnio skaičiavimų apimties kitimas prie tam tikrų VVL lygių didinant viršūnių skaičių. Dijkstros algoritmo skaičiavimai nepriklauso nuo VVL, todėl jie pavaizduoti atskiru grafiku.




**32 pav. 1 iteracijos skaičiavimų kiekio priklausomybė nuo viršūnių skaičiaus**

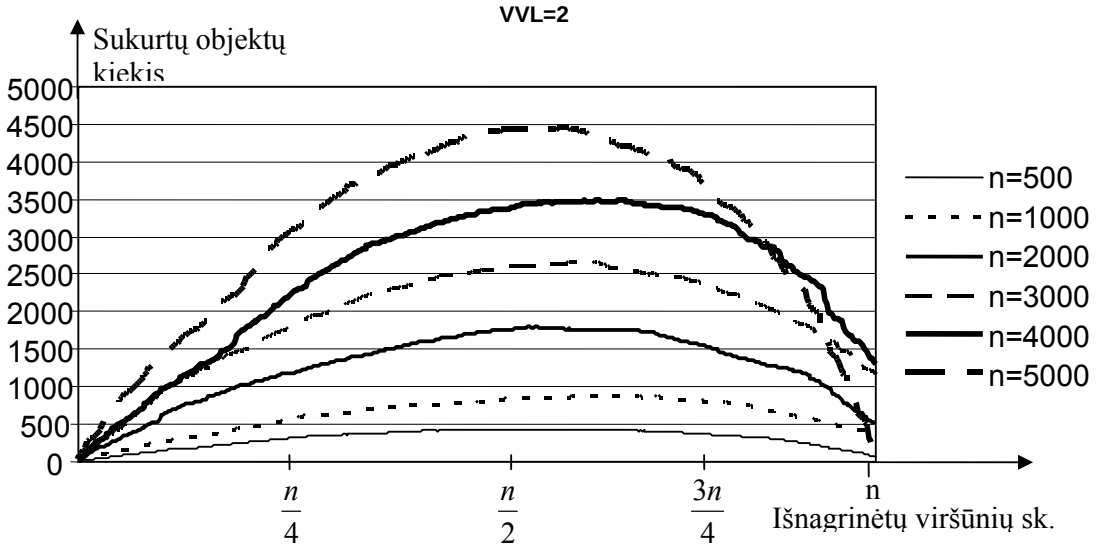



**33 pav. Bendro skaičiavimų kiekio priklausomybė nuo viršūnių skaičiaus**

Kaip matome, 33 pav. iki 1000 viršūnių turinčiuose grafuose algoritmų skaičiavimų apimtys praktiškai nesiskiria. Mazgų skaičiui esant didesniam už 2000, ima ryškėti patobulinto algoritmo pranašumas. Šis pranašumas ypač gerai pastebimas grafuose, kurių VVL mažesnis arba lygus 50.

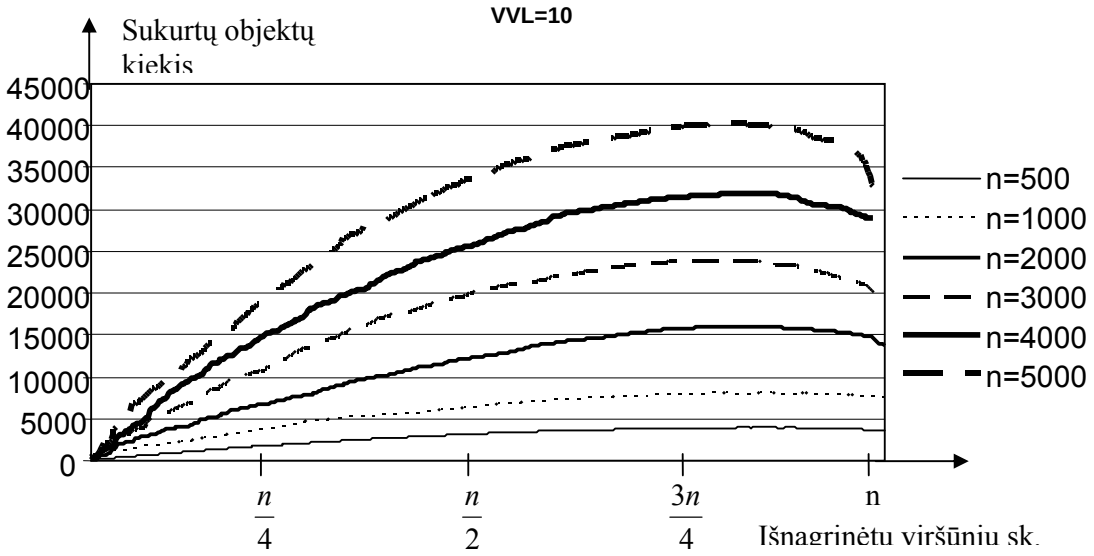
## 4.4 Naudojamos atminties priklausomybių grafikai

Šiame skyriuje pateikiami grafikai, kurie nusako užimamos atminties priklausomybes nuo grafo VVL ir grafo viršūnių skaičiaus n.

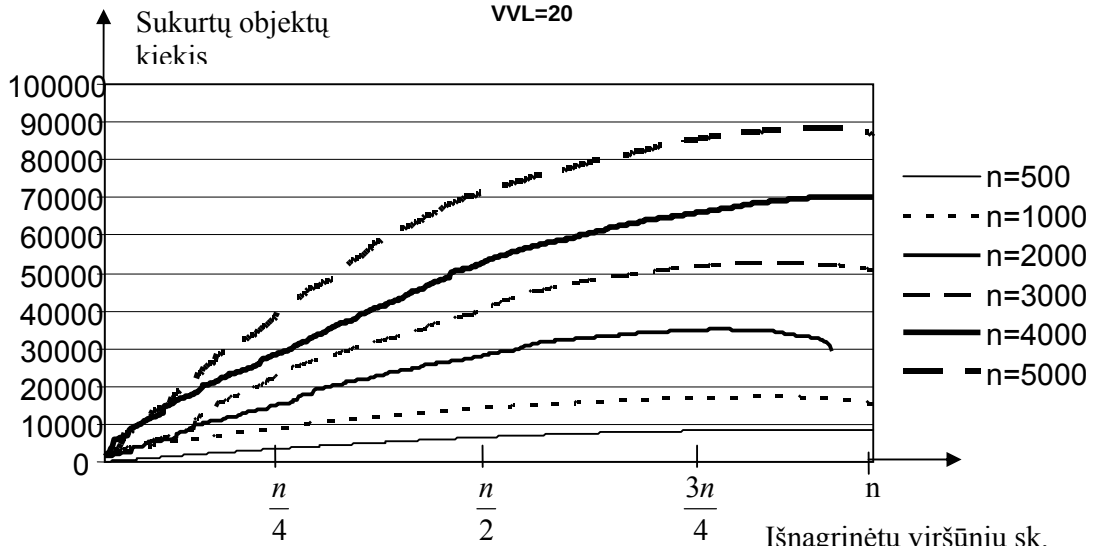



**34 pav. Išskirtos atminties kiekio priklausomybė nuo sprendimo progreso, kai VVL=2**

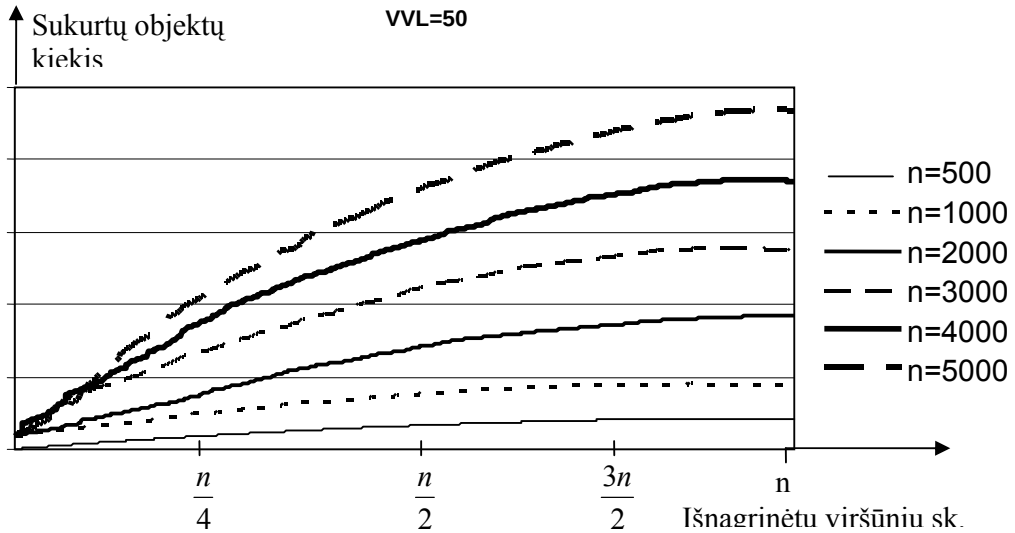
34 pav. pavaizduotas dinaminiame lygyje išskirtos atminties kiekio kitimas, kai VVL=2. Maksimalus išskirtos atminties kiekis pastebimas įpusėjus uždavinį. Uždavinio pradžioje ir pabaigoje išskirtos atminties kiekis yra žymiai mažesnis.




**35 pav. Išskirtos atminties kiekio priklausomybė nuo sprendimo progreso, kai VVL=10**

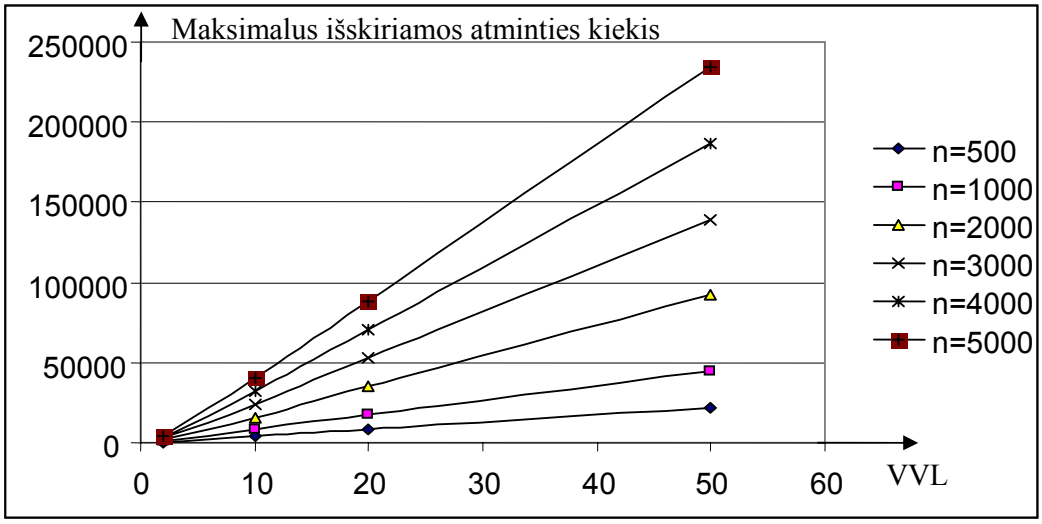



**36 pav. Išskirtos atminties kiekio priklausomybė nuo sprendimo progreso, kai VVL=20**

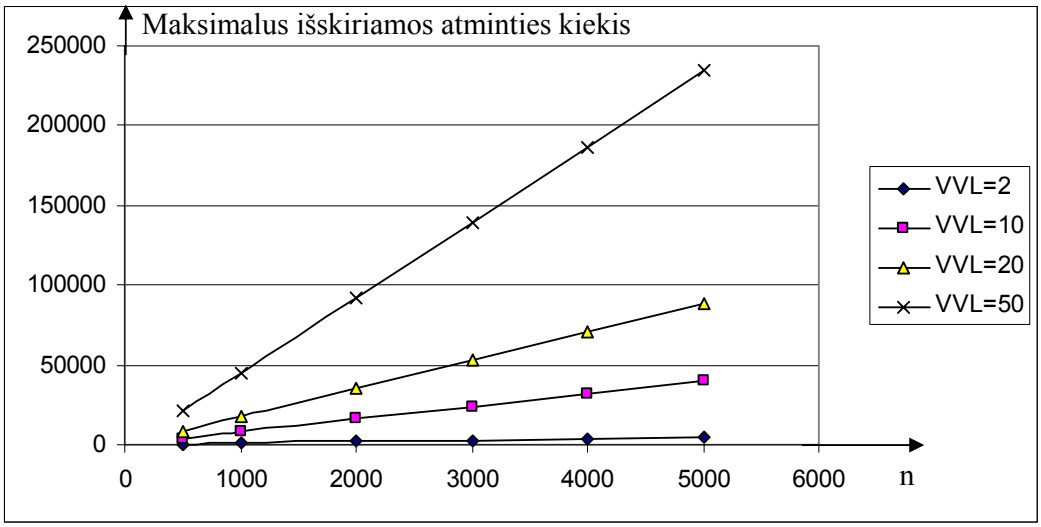



**37 pav. Išskirtos atminties kiekio priklausomybė nuo sprendimo progreso, kai VVL=50**

35, 36, 37 priklausomybėse pastebime, kad išskirtos atminties kiekis išsprendus daugiau nei pusę uždavinio toliau didėja. Taip pat pastebime, kad uždavinio progresui artėjant prie pabaigos, išskirtos atminties kiekis ženkliai nemažėja. Tačiau dėl tinkamai parinktų duomenų struktūrų net ir esant tokiam dideliam duomenų kiekiui, skaičiavimų, lyginant su Dijkstros algoritmu, atliekama santykinai nedaug.




**38 pav. Maksimaliai išskirtos atminties kiekio priklausomybė nuo VVL prie įvairių viršūnių skaičiaus n lygių**

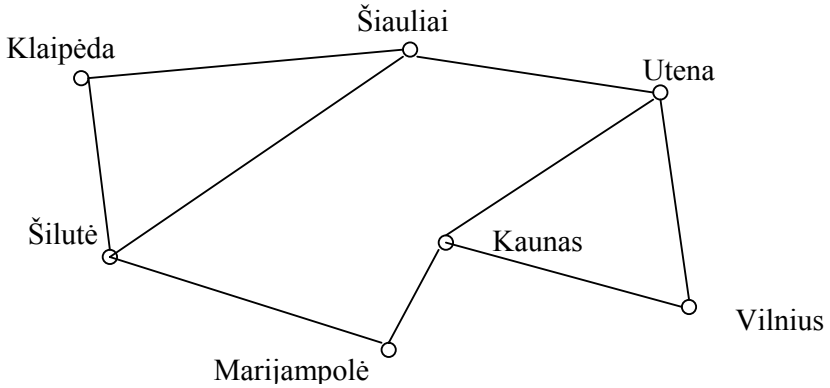



**39 pav. Maksimaliai išskirtos atminties kiekio priklausomybė nuo viršūnių skaičiaus n prie įvairių VVL lygių**

38 ir 39 pav. pavaizduotos dinaminiame lygyje maksimaliai išskiriamos atminties priklausomybės nuo VVL ir nuo viršūnių skaičiaus n. Kaip matome, maksimaliai išskiriama atmintis tiesiškai priklauso tiek nuo VVL, tiek nuo viršūnių skaičiaus n. Iš to seka, kad norėdami išspręsti x kartų didesnį uždavinį, mums reikės atminties $c \cdot x$ kartų daugiau, nei spręstam mažesniam uždaviniui; čia c – konstanta, priklausanti nuo vieno išskiriamo objekto dydžio ir algoritmo realizacijos.

## 4.5 Eksperimentinis apkrautumo optimizavimas SDH tinkle

Šiame skyriuje pademonstruosime nedidelį srautų optimizavimo SDH tinkle pavyzdį. Tarkime turime SDH tinklą (pav. 40):




**40 pav. SDH tinklo pavyzdys**

Panaudokime mazgų pavadinimų sutrumpinimus: mazgą „Klaipėda“ žymėsime A, „Šiaulius“ – B, „Šilutę“ – C, „Marijampolę“ – D, „Kauną“ – E, „Uteną“ – F, „Vilnių“ – G.

Šio tinklo instaliuoti resursai aprašyti gretimumo struktūros tipo matrica $\{r_{ij}|i=\overline{1,n}\,;\, j=\overline{1,n}\}$, kurios kiekvienas elementas $r_{ij}$ išreiškia instaliuotų VC-12 srautų kiekį linijoje tarp mazgų i ir j (žr 2.2 skyrių):

**3 lentelė. Matricos $r_{ij}$ pavyzdys**

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| A | 0 | 20 | 25 | 0 | 0 | 0 | 0 |
| B | 20 | 0 | 10 | 0 | 0 | 30 | 0 |
| C | 25 | 10 | 0 | 30 | 0 | 0 | 0 |
| D | 0 | 0 | 30 | 0 | 10 | 0 | 0 |
| E | 0 | 0 | 0 | 10 | 0 | 20 | 40 |
| F | 0 | 30 | 0 | 0 | 20 | 0 | 15 |
| G | 0 | 0 | 0 | 0 | 40 | 15 | 0 |

Tinklo panaudoti resursai aprašomi matrica $\{p_{ij}|i=\overline{1,n}\,;\, j=\overline{1,n}\}$, kurios kiekvienas elementas $p_{ij}$ reiškia panaudotų VC-12 srautų kiekį linijoje tarp mazgų i ir j.

**4 lentelė. Matricos $p_{ij}$ pavyzdys**

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| A | 0 | 8 | 15 | 0 | 0 | 0 | 0 |
| B | 8 | 0 | 5 | 0 | 0 | 15 | 0 |
| C | 15 | 5 | 0 | 15 | 0 | 0 | 0 |
| D | 0 | 0 | 15 | 0 | 3 | 0 | 0 |
| E | 0 | 0 | 0 | 3 | 0 | 6 | 10 |
| F | 0 | 15 | 0 | 0 | 6 | 0 | 5 |
| G | 0 | 0 | 0 | 0 | 10 | 5 | 0 |

Tinklo operatorius užduoda pageidaujamą rezervą kiekvienai linijai, kuris aprašomas matrica $\{al_{ij}|i=\overline{1,n}\,;\, j=\overline{1,n}\}$, kurios kiekvienas elementas $al_{ij}$ – tai rezervuotų VC-12 srautų kiekis linijoje tarp mazgų i ir j.

**5 lentelė. Matricos $al_{ij}$ pavyzdys**

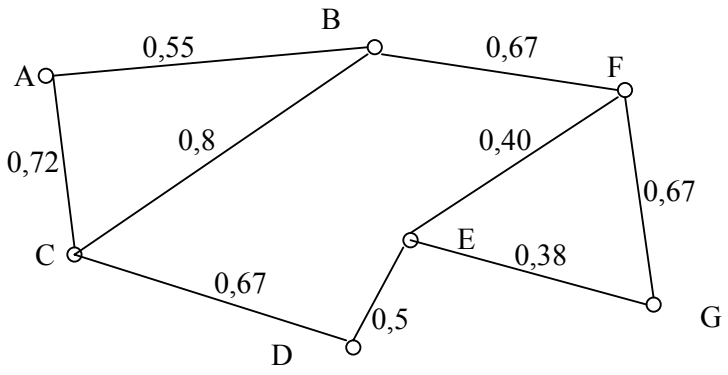
| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| A | 0 | 3 | 3 | 0 | 0 | 0 | 0 |
| B | 3 | 0 | 3 | 0 | 0 | 5 | 0 |
| C | 3 | 3 | 0 | 5 | 0 | 0 | 0 |
| D | 0 | 0 | 5 | 0 | 2 | 0 | 0 |
| E | 0 | 0 | 0 | 2 | 0 | 2 | 5 |
| F | 0 | 5 | 0 | 0 | 2 | 0 | 5 |
| G | 0 | 0 | 0 | 0 | 5 | 5 | 0 |

Pagal formulę (5) apskaičiuojame matricą $\{g_{ij} | i = \overline{1,n}\ ;\ j = \overline{1,n}\}$, vaizduojančią linijų, jungiančių bet kuriuos du mazgus i ir j, santykinį apkrautumą:

**6 lentelė. Matricos $g_{ij}$ pavyzdys**

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| A | 0,00 | 0,55 | 0,72 | ∞ | ∞ | ∞ | ∞ |
| B | 0,55 | 0,00 | 0,80 | ∞ | ∞ | 0,67 | ∞ |
| C | 0,72 | 0,80 | 0,00 | 0,67 | ∞ | ∞ | ∞ |
| D | ∞ | ∞ | 0,67 | 0,00 | 0,50 | ∞ | ∞ |
| E | ∞ | ∞ | ∞ | 0,50 | 0,00 | 0,40 | 0,38 |
| F | ∞ | 0,67 | ∞ | 0,00 | 0,40 | 0,00 | 0,67 |
| G | ∞ | ∞ | ∞ | ∞ | 0,38 | 0,67 | 0,00 |

Pagal 6 lentelėje pateiktą matricą pradinis linijų apkrautumo grafas pavaizduotas 41pav.




**41 pav. SDH tinklo linijų apkrautumo grafo pavyzdys**

Tarkime, mums keliamas uždavinys optimaliai maršrutizuoti VC-2 srautą nuo mazgo A iki mazgo G. Uždavinį sprendžiame skaidydami srautą VC-2 į 3 VC-12 srautus ir maršrutizuojame juos atskirai. Naudodami bet kurį trumpiausių kelių tarp dviejų viršūnių radimo algoritmą gauname, kad trumpiausias kelias tarp viršūnių A ir G yra šis: $\mu_1 = \{(A, B), (B, F), (F, G)\}$. Jo briaunų svorių suma lygi 1,89. Per rastą kelią vedame pirmąjį VC-12 srautą. Po šio veiksmo pasikeičia panaudotų resursų kiekis linijose, per kuriuos šis srautas buvo išvestas, dėl to perskaičiuojame matricas $\{p_{ij}\}$ ir $\{g_{ij}\}$:

**6 lentelė. Matricos $p_{ij}$ ir $g_{ij}$ po pirmo skaičiavimų žingsnio**

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| A | 0 | 9 | 15 | 0 | 0 | 0 | 0 |
| B | 9 | 0 | 5 | 0 | 0 | 16 | 0 |
| C | 15 | 5 | 0 | 15 | 0 | 0 | 0 |
| D | 0 | 0 | 15 | 0 | 3 | 0 | 0 |
| E | 0 | 0 | 0 | 3 | 0 | 6 | 10 |
| F | 0 | 16 | 0 | 0 | 6 | 0 | 6 |
| G | 0 | 0 | 0 | 0 | 10 | 6 | 0 |

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| A | 0,00 | 0,60 | 0,72 | ∞ | ∞ | ∞ | ∞ |
| B | 0,60 | 0,00 | 0,80 | 0,00 | ∞ | 0,70 | ∞ |
| C | 0,72 | 0,80 | 0,00 | 0,67 | ∞ | ∞ | ∞ |
| D | ∞ | ∞ | 0,67 | 0,00 | 0,50 | ∞ | ∞ |
| E | ∞ | ∞ | ∞ | 0,50 | 0,00 | 0,40 | 0,38 |
| F | ∞ | 0,70 | ∞ | ∞ | 0,40 | 0,00 | 0,73 |
| G | ∞ | ∞ | ∞ | ∞ | 0,38 | 0,73 | 0,00 |

Antrajam VC-12 srautui, analogiškai kaip ir pirmame žingsnyje matricoje $\{g_{ij}\}$ ieškome trumpiausio kelio tarp viršūnių A ir G. Rastas kelias taip pat sutampa su rastuoju pirmame žingsnyje: $\mu_2 = \{(A, B), (B, F), (F, G)\}$. Tačiau jo svoris jau yra didesnis: 2,03. Antrąjį VC-12 srautą vedame per rastą optimalų maršrutą ir perskaičiuojame matricas $\{p_{ij}\}$ ir $\{g_{ij}\}$:

**7 lentelė. Matricos $p_{ij}$ ir $g_{ij}$ po antrojo skaičiavimų žingsnio**

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| A | 0 | 10 | 15 | 0 | 0 | 0 | 0 |
| B | 10 | 0 | 5 | 0 | 0 | 17 | 0 |
| C | 15 | 5 | 0 | 15 | 0 | 0 | 0 |
| D | 0 | 0 | 15 | 0 | 3 | 0 | 0 |
| E | 0 | 0 | 0 | 3 | 0 | 6 | 10 |
| F | 0 | 17 | 0 | 0 | 6 | 0 | 7 |
| G | 0 | 0 | 0 | 0 | 10 | 7 | 0 |

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| A | 0,00 | 0,65 | 0,72 | ∞ | ∞ | ∞ | ∞ |
| B | 0,65 | 0,00 | 0,80 | ∞ | ∞ | 0,73 | ∞ |
| C | 0,72 | 0,80 | 0,00 | 0,67 | ∞ | ∞ | ∞ |
| D | ∞ | ∞ | 0,67 | 0,00 | 0,50 | ∞ | ∞ |
| E | ∞ | ∞ | ∞ | 0,50 | 0,00 | 0,40 | 0,38 |
| F | ∞ | 0,73 | ∞ | ∞ | 0,40 | 0,00 | 0,80 |
| G | ∞ | ∞ | ∞ | ∞ | 0,38 | 0,80 | 0,00 |

Trečiajame žingsnyje optimalus maršrutas tarp A ir G pasikeičia: $\mu_3 = \{(A, B), (B, F), (F, E), (E, G)\}$. Šio maršruto svoris: 2,16.

Taigi, išsprendę uždavinį, gavome, kad srautas VC-2, kurį reikėjo išvesti tarp viršūnių A ir G, eina šiuo keliu: $\mu_{o1} = \{(A, B), (B, F)\}$. Mazge F jis skaidomas į 3 VC-12 srautus, iš kurių 2 maršrutizuojami keliu $\mu_{o21} = \{(F, G)\}$, o 1 – keliu $\mu_{o22} = \{(F, E), (E, G)\}$. Mazge G srautai vėl sujungiami į vieną VC-2 srautą. Pagal 2.3 skyrių šis maršrutas tolygaus tinklo apkrautumo požiūriu laikomas optimaliu.

## 4.6 Išvados

- Algoritmų kokybę tikslinga įvertinti ne laiku, o veiksmų kiekiu
- Esant nedideliam VVL patobulinto algoritmo skaičiavimų apimtis artima $\Theta(n)$
- Kai viršūnių skaičius n > 500 ir VVL < 45, patobulintas trumpiausio kelio tarp dviejų viršūnių algoritmas uždavinį sprendžia mažesne skaičiavimų apimtimi nei tradicinis Dijkstros algoritmas
- Eksperimentinis tyrimas patvirtina teorinius samprotavimus

# 5 IŠVADOS

1. SDH tinklo apkrautumo matematinis modelis yra grafas. Grafo briaunų, vaizduojančių SDH linijų apkrautumą, svoriai gali būti apibrėžti kaip žemiausio hierarchinio lygio nagrinėjamų srautų dedamųjų VC-11 suma.
2. Žinomų grafų teorijos trumpiausio kelio tarp dviejų viršūnių paieškos algoritmų, naudojamų SDH tinklo apkrautumo optimizavimo uždaviniams spręsti, vykdymo laikas yra per didelis. Kadangi SDH tinklai pasižymi nedideliu VVL, dėl to buvo sukurtas šią savybę įvertinantis algoritmas, randantis sprendinį per trumpesnį laiką.
3. Ištyrus duomenų struktūras, taikomas Dijkstros algoritmui, pasiektas skaičiavimų sudėtingumas blogiausiu atveju neviršijantis standartinio šio algoritmo sudėtingumo $\Theta(n^2)$, o geriausiu atveju skaičiavimų sudėtingumas vykdant uždavinį didėja nuo $\Theta(n)$ iki $\Theta(n^2)$.
4. Eksperimentiškai nustatyti skaičiavimo apimties priklausomybės nuo VVL lūžio taškai, iki kurių patobulintas algoritmas atlieka mažiau skaičiavimų nei tradicinis Dijkstros algoritmas. Rastas maksimalus atminties kiekis, reikalingas išspręsti n mazgų ir m linijų SDH tinklo apkrautumo optimizavimo uždavinį.
5. Kai viršūnių skaičius $n > 500$ ir $VVL < 45$, patobulintas trumpiausio kelio tarp dviejų viršūnių algoritmas uždavinį sprendžia mažesne skaičiavimų apimtimi nei tradicinis Dijkstros algoritmas

# 6 LITERATŪROS SĄRAŠAS

1. Plukas, K., Mačikėnas, E., Jarašiūnienė, B., Mikuckienė, I. Taikomoji diskrečioji matematika. Kaunas: Technologija, 2003. ISBN 9955-09-031-6.

2. Липский, В. Комбинаторика для програмистов. Москва: Мир, 1998. ISBN 5-03-000979-5.

3. Plėštys, R., Kišonas V., Šinickas, D. Regionų telekomunikacijų resursų įvertinimo metodika // Regionų plėtra-2002: tarptautinės konferencijos pranešimų medžiaga. ISBN 9955-09-275-0.-Kaunas: Technologija, 2002.

4. Vidžiūnas, A. Delphi 6. Programavimas ir vaizdiniai komponentai. Kaunas: Smaltija, 2002. ISBN 9986-965-95-0.

5. ITU-T Rec. G.707. Network node interface for the synchronous digital hierarchy (SDH). Geneve: ITU, 1996. - P.1-129.

# 7 TERMINŲ IR SANTRUMPŲ ŽODYNĖLIS

**Tankinimas (*Multiplexing*)** - procesas, kai žemesnės eilės trakto signalas derinamas prie aukštesnės eilės trakto signalo arba kai aukštesnės eilės trakto signalas derinamas prie tankinimo sekcijos.

**VVL** – vidutinis viršūnių laipsnis.

**SDH – *(synchronous digital hierarchy).*** Skaitmeninė sinchroninė hierarchija.

**PDH** – pleziochroninė skaitmeninė hierarchija

**STM-256** – 40Gb/s SDH srautas

**STM-64** – 10Gb/s SDH srautas

**STM-16** – 2,5Gb/s SDH srautas

**STM-4** – 622Mb/s SDH srautas

**STM-1** – 150Mb/s PDH srautas

**VC-3** – 34Mb/s PDH srautas

**VC-2** – 6Mb/s PDH srautas

**VC-11**- 1,5Mb/s nedalomas PDH srautas

**VC-12** – 2Mb/s nedalomas PDH srautas

# 8 SUMMARY

The software currently used by SDH net operators does not guarantee the automatic generation of routes between the selected nodes. This is the reason why the operators spend a lot of time on designing the nets manually. Despite all the time spent on designing, errors are still made which are the causes of net overload and connection disorders.

It is necessary to convert the measuring units of SDH resources from a tree type structure to real numbers, because the structure of the analyzed object's resource units is hierarchical. After evaluation of these units the solution of this problem can be found in theory of graphs.

It is natural that given these data structures a special method is needed which would allow determining these structures and finding specific weight unit of edges of the graphs with which the problem is solved. The complexity of graph route composition algorithms is not satisfactory because the scope of real systems will be big enough. The amount of net nodes can exceed 500 and the amount of lines connecting them can be more than 5000.

It is not very difficult to implement the automatic generation of routes in principle. The knowledge of graph theory can be used to do that. But in solving real problems with the best known methods of graph theory the time of solution is not acceptable because of the big scope of the problem. It is possible to reduce the time of solution by increasing the efficiency of the system that solves the problem. The need for more simple algorithms from the point of view of calculation scope and need for the evaluation of their complexity have emerged because of these reasons. The descriptions of algorithms are given and their complexities are explored in this work.

The problems solved are:

- To determine methodic for unambiguous evaluation of SDH resources;
- To propose a methodic for graphs that reflect the relative load of SDH net lines. This problem determines a criteria for SDH net resource optimization – equal relative net load;
- To analyze the weak characteristics of currently used algorithms and to evaluate the complexity of their calculations;
- To propose data structures using which particular algorithm operations would be performed with smaller scope of calculations;
- To propose an algorithm which would be specially adapted for big graphs with relatively small average node degree.

# 9 PRIEDAI

## 9.1 Patobulinto algoritmo realizacija

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Grids, ComCtrls;

const n=500;

type
  TMtr = array[1..n, 1..n] of integer;

  TForm1 = class(TForm)
    Button1: TButton;
    StringGrid1: TStringGrid;
    Button2: TButton;
    StringGrid2: TStringGrid;
    Button3: TButton;
    Button4: TButton;
    ProgressBar1: TProgressBar;
    Edit1: TEdit;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Dijkstra(var M: tmtr);
    procedure generatemasp;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  link = ^rec;

  rec=record
        nr: integer;
        len: integer;
        next: link;
```

```
  end;

  TMas = array[1..n] of rec;

var
  Form1: TForm1;
  ap, ag: integer;
  mas, mas_p: TMas;
  choice: TMas; nc: integer;
  mtr, mtr1: TMtr;
  check: array[1..n] of boolean;
  dc, dc1, dc2: integer;

implementation
{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var x, y, i, s: integer;
p:link;
begin
      for x := 1 to n do begin
            for y := 1 to n do begin
                  if x=y then mtr[x,y]:=0 else mtr[x, y] := n*n;
            end;
      end;

      randomize;

      for i := 1 to (strtoint(Edit1.Text)) do begin
            x := trunc(random(n-1)+1);
            y := trunc(random(n-1)+1);
            while x = y do y := trunc(random(n-1)+1);
            s := trunc(random(n{*10})+1);
            mtr[x, y] := s;
            mtr[y, x] := s;
      end;
      mas[1].nr:=1;
      mas[1].next:=nil;
      for i := 2 to n do begin
            if (mtr[1, i]>0) and (mtr[1,i]<n*n) then begin
                  p := new(link);
                  p.nr := i;
                  p.len:=mtr[1, i];
                  p.next := mas[1].next;
                  mas[1].next:=p;
            end;
      end;
```

```
        for x := 1 to n do begin
            mas_p[x].nr := x;
            mas_p[x].next := nil;
            for y := 1 to n do begin
                if (mtr[x, y]>0) and (mtr[x,y]<n*n) then begin
                    p := new(link);
                    p.nr := y;
                    p.len:=mtr[x, y];
                    p.next := mas_p[x].next;
                    mas_p[x].next:=p;
                end;
            end;
        end;

        Check[1] := True;
        for i := 2 to n do Check[i]:=false;
        ap:=1;
        ag:=1;
end;


function Sort(Choice: TMas; nc: integer): TMas;
var i, j: integer;
      r: rec;
begin
        for i := 1 to nc do begin
            for j := 1 to nc do begin
                if (Choice[i].len<Choice[j].len) then begin
                    r := Choice[i];
                    Choice[i]:=Choice[j];
                    Choice[j]:=r;
                    dc1 := dc1 + 1;
                end;
            end;
        end;
        Result := Choice;
end;


procedure TForm1.Button2Click(Sender: TObject);
var
        p: link;
        done: boolean;
        i: integer;
begin
        while (ap<=ag) do begin
```

```
            if mas[ap].next<>nil then begin
                if not Check[mas[ap].next.nr] then begin
                    ag := ag + 1;
                    mas[ag].nr := mas[ap].next.nr;
                    mas[ag].next := mas_p[mas[ag].nr].next;
                    done := false;
                    while (mas[ag].next<>nil) and (not done) do begin
                        dc1 := dc1 + 1;
                        if Check[mas[ag].next.nr] then mas[ag].next:=mas[ag].next.next else done := true;
                    end;
                    p := mas[ag].next;
                    if p<>nil then
                        while p.next <> nil do begin
                            dc1 := dc1 + 1;
                            if Check[p.next.nr] then p.next := p.next.next else p := p.next;
                        end;
                    nc := 0;
                    dc1 := dc1 + ag-ap;

                    for i := ap to ag do begin
                        p := mas[i].next;

                        if mtr[mas[i].nr, mas[ag].nr]<n*n then begin
                            nc := nc + 1;
                            choice[nc].nr := mas[i].nr;
                            choice[nc].len := mtr[mas[i].nr, mas[ag].nr]+mtr[1, mas[i].nr];
                        end;

                    end;
                    if nc > 1 then Choice := Sort(Choice, nc);
                    if (Choice[1].len<mtr[1, mas[ag].nr]) then begin
                        mtr[1, mas[ag].nr] := Choice[1].len;
                        mtr[mas[ag].nr, 1] := Choice[1].len;
                    end;
                    Check[mas[ag].nr] := true;
                end else mas[ap].next := mas[ap].next.next;
            end else ap := ap + 1;
        end;
    end;

    end.
```

## 9.2 SDH tinklo srautų maršrutizavimo programinė įranga