KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

Saulius Lazaravičius

TELEKOMUNIKACIJŲ PRIEIGOS TINKLO OPTIMIZAVIMO UŽDAVINIŲ ANALIZĖ IR REALIZACIJA

Magistro darbas

Vadovas
doc. dr. N. Listopadskis

KAUNAS, 2007

KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
prof. dr. J.Rimas
2007 06 06

TELEKOMUNIKACIJŲ PRIEIGOS TINKLO OPTIMIZAVIMO UŽDAVINIŲ ANALIZĖ IR REALIZACIJA

Taikomosios matematikos magistro baigiamasis darbas

Vadovas
( ) doc. dr. N. Listopadskis
2007 06 01

Recenzentas
( ) dr. V. Grimaila
2007 06 05

Atliko
FMMM 5 gr. stud.
( ) S. Lazaravičius
2007 05 25

KAUNAS, 2007

KVALIFIKACINĖ KOMISIJA

Pirmininkas: Leonas Saulis, habil. dr., Vilniaus Gedimino technikos universiteto profesorius.

Sekretorius: Eimutis Valakevičius, docentas.

Nariai: Algimantas Jonas Aksomaitis, profesorius,

Arūnas Barauskas, dr., UAB „Elsis“ generalinio direktoriaus pavaduotojas,

Vytautas Janilionis, docentas,

Zenonas Navickas, profesorius,

Vidmantas Povilas Pekarskas, profesorius,

Rimantas Rudzkis, habil.dr., banko „NORD/LB“ vyriausiasis analitikas.

Lazaravičius S. Telekomunikacijų prieigos tinklo optimizavimo uždavinių analizė ir realizacija: Taikomosios matematikos magistro darbas / darbo vadovas doc. dr. N. Listopadskis, Taikomosios matematikos katedra, Fundamentaliųjų mokslų fakultetas, Kauno technologijos universitetas. – Kaunas, 2007. – 98 p.

SANTRAUKA

Darbo tikslas – sukurti bendrą prieigos tinklo modeliavimo metodiką bei jos programinę realizaciją, atitinkančią šiuos reikalavimus:

- n užduotų prieigos tinklo parametrų reikšmių optimalus nustatymas pagal m užduotų prieigos tinklo kokybės apribojimų, kai $n \geq 1$, o $m \geq 0$;
- optimalus stočių koordinačių nustatymas mobiliojo telefono ryšio tinklui;
- optimalus stočių koordinačių nustatymas laidinio telefono ryšio tinklui;

Darbo pradžioje apžvelgiamos telekomunikacijų sektoriaus užduotys, kurios gali būti sprendžiamos kombinatorinio optimizavimo metodais. Taipogi pristatomi ir suklasifikuojami galimi šių užduočių sprendimo metodai.

Tiriamojoje darbo dalyje pristatomas daugiaparametrinis prieigos tinklo optimizavimo algoritmas integruotas su stočių išdėstymo algoritmais. Stočių išdėstymui pateikiami du meta-euristiniai algoritmai:

- Skruzdžių kolonijos algoritmas, papildytas lokalios paieškos procedūra;
- Genetinis algoritmas, papildytas lokalios paieškos procedūra.

Minėtų algoritmų realizacijos skirstomos pagal šias prieigos tinklo ryšio topologijas:

- Mobiliojo telefono ryšio tinklui;
- Fiksuoto telefono ryšio tinklui.

Esminiai darbe pasiekti rezultatai:

- Sukurta universali metodika, leidžianti kurti realius prieigos tinklo modelius;
- Sukurta šios metodikos programinė realizacija.

Darbe nagrinėjamų uždavinių ir algoritmų pagrindu buvo paskelbti ir pristatyti šie straipsniai:

- „Prieigos tinklo parametrų optimalaus parinkimo algoritmas“, Matematika ir matematikos dėstymas, 2006;
- “Teritorijos padengimo uždavinio analizė”, VI studentų konferencija, KTU, 2006;
- “Stočių išdėstymo algoritmo analizė”, Lietuvos matematikų draugijos XLVII konferencijoje, 2006.

Taip pat pateiktas straipsnis „Fiksuoto telefono ryšio tinklo stočių išdėstymo algoritmas“ Lietuvos matematikų draugijos XLVIII konferencijai.

Lazaravičius S. Analysis and realization of telecommunication network approach optimization algorithms: Master‘s work in applied mathematics / supervisor dr. assoc. prof. N. Listopadskis; Department of Applied mathematics, Faculty of Fundamental Sciences, Kaunas University of Technology. – Kaunas, 2007. – 98 p.

SUMMARY

The objective of this work is creation of telecommunication network approach algorithm and its realization. The created algorithm must fulfill following requirements:

- optimal values evaluation of n given network approach parameters with m given network approach quality constrains, where $n \geq 1$, o $m \geq 0$;
- optimal solution for transmitters placement problem in mobile phone network;
- optimal solution for transmitters placement problem in fixed phone network;

In the beginning of this paper we present a set of telecommunication segment problems which can be solved using combinatorial optimization methods. Also we present a set of combinatorial optimization methods which can be used for solving these problems. Finally we present a graphical classification of analyzed problems and connect it with algorithms which are capable for solving it.

In the research part of this paper we present a multi parametric network approach optimization algorithm united with algorithms for placing transmitters. Next we present two Meta heuristics based optimization algorithms:

- Ant Colony Optimization algorithm with local search procedure;
- Genetic algorithm with local search procedure.

The realization of these two algorithms depends on the topology of the network approach being analyzed. In this paper we analyze two most common types of network approaches:

- Mobile phone network approach;
- Fixed phone network approach.

The two main achievements of this work:

- Creation of universal methodology, which allows to create a network approach model;
- Creation of software which implements methodology mentioned.

Based on the problems and algorithms analyzed in this paper these researches were presented:

- „Algorithm for optimal network approach parameters selection“, Mathematics and its lecturing, 2006;
- “The Analysis of Territory Covering Problem”, VI student conference, KTU, 2006;
- “The Analysis of ACO Algorithm for Transmitters Locating Problem”, XLVII conference of Lithuanian Mathematicians Association, 2006.

Also we submitted a research called “Algorithm for a Fixed Charge Telephone Network Approach Transmitters Placing Problem” to the XLVIII conference of Lithuanian Mathematicians Association.

TURINYS

PAVEIKSLŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

1. ĮVADAS

Kombinatorinis optimizavimas - tai procesas, kurio metu sprendžiami tokie optimizavimo uždaviniai, kur galimi sprendiniai yra arba diskretūs dydžiai, arba gali būti pakeisti diskrečiais. Proceso metu siekiama surasti užduoties sąlygą tenkinantį sprendinį. Šios matematikos mokslo srities ištakos siejamos su tiesinio programavimo teorija ir yra glaudžiai susijusios su diskrečiąja matematika, tikimybių teorija, įvairiais kompiuterių mokslais. Tiksliai išspręsti kombinatorinio optimizavimo uždavinį yra labai sunku, tačiau atskirų uždavinių sprendimui yra sukurta nemažai metodų, kurie gana kokybiškai randa ieškomus sprendinius. Šiais laikais, kai technologijos sparčiai vystosi, atsirado daugiau galimybių spręsti kombinatorinio optimizavimo problemas.

Apžvelgti visus kombinatorinio optimizavimo uždavinius yra sunku. Paminėsime, kad ši matematikos mokslo šaka leidžia spręsti įvairius pjaustymo, pakavimo, darbų įrengimams priskyrimo, tvarkaraščių sudarymo ir telekomunikacijų sektoriaus problemas. Vystantis technologijoms ir didėjant telekomunikacijų reikšmei, visuomenėje atsirado didelis poreikis spręsti su tuo susijusias problemas: teritorijos padengimas siųstuvais mobiliojo ryšio tinkle, optimalaus dažnių juostos naudojimo stotyse arba fiksuoto ryšio abonentų sujungimas į tinklą. Visuose minėtuose uždaviniuose esminė problema yra ta, kad vartotojams reikia teikti kokybiškas ryšio paslaugas, optimaliai paskirstant kompanijos resursus. T. y. reikia tiekti ne mažesnes nei nustatytos kokybės paslaugas minimizuojant tų paslaugų teikimo kaštus. Pavyzdžiui, galima teritorijoje pristatyti daug siųstuvų ir ryšys bus puikus, bet tai brangiai kainuoja. Galima kiekvienam susijungimui su stotimi priskirti reikšmingai dažniu besiskiriantį kanalą, bet tuomet reikės plačios dažnių juostos, o tai yra problematiška. Minėtais atvejais, pritaikę kombinatorinio optimizavimo metodus, galime iš dalies arba visiškai išspręsti šias problemas. Siųstuvus pastatyti taip, kad užtektų mažesnio jų kiekio per daug nesumažinant ryšio kokybės. Jei vartotojai yra pakankamai toli vienas nuo kito, jie komunikuoti gali kanalais, kurių dažniai vienodi, o arti vienas kito esantiems bus priskiriami skirtingi dažniai. Vadinasi, galėsime teikti kokybiškas ryšio paslaugas, tačiau tai darysime operuodami siauresne dažnių juosta.

1.1 DARBO TIKSLAS IR UŽDUOTYS

Šiame darbe kombinatorinio optimizavimo metodais spręsime šias telekomunikacijų sektoriaus užduotis:

- optimalaus prieigos tinklo parametrų parinkimo problemą;
- optimalaus stočių išdėstymo problemą mobiliojo telefono ryšio tinklui;
- optimalaus stočių išdėstymo problemą fiksuoto telefono ryšio tinklui;

Darbo tikslas – sukurti bendrą prieigos tinklo modeliavimo metodiką bei jos programinę realizaciją, atitinkančią šiuos reikalavimus:

- n užduotų prieigos tinklo parametrų reikšmių optimalus nustatymas pagal m užduotų prieigos tinklo kokybės apribojimų, kai $n \geq 1$, o $m \geq 0$;
- optimalus stočių koordinačių nustatymas mobiliojo telefono ryšio tinklui;
- optimalus stočių koordinačių nustatymas laidinio telefono ryšio tinklui.

1.2 UŽDUOČIŲ SPRENDIMO BŪDAI

Darbo tikslo įgyvendinimui, pristatysime šiuos algoritmus:

1) Modifikuotą daugiaparametrinį prieigos tinklo optimizavimo algoritmą, pateiktą literatūroje [6]. Minėtą algoritmą modifikuosime dviem aspektais:

- optimalių parametrų reikšmių parinkimo aspektu. T. y. modifikuoto algoritmo surastos parametrų reikšmės bus geresnės nei literatūroje [6] pateikiamo daugiaparametrinio prieigos tinklo optimizavimo algoritmo reikšmės tinklo kaštų dydžio prasme;
- stočių išdėstymo aspektu. T. y. modifikuotas algoritmas bus pritaikytas integracijai su stočių išdėstymo algoritmais;

2) Modifikuotą genetinį algoritmą mobiliojo telefono ryšio tinklo stočių išdėstymui pateiktą literatūroje [1]. Algoritmą modifikuosime lokalios paieškos procedūra, dėl to algoritmo surastos stočių dislokacijos vietos bus geresnės nei literatūroje [1] pateikiamo algoritmo tinklo kaštų dydžio prasme.

3) Skruzdžių kolonijos algoritmą su lokalios paieškos procedūra mobiliojo telefono ryšio tinklo stočių išdėstymui, pateiktą literatūroje [7].

4) Modifikuotą genetinį algoritmą fiksuoto telefono ryšio tinklo stočių išdėstymui. Čia pateiksime naują genetinio algoritmo interpretaciją, papildytą lokalios paieškos procedūra, dėl to algoritmo surastos stočių dislokacijos vietos bus geresnės nei klasikinio genetinio algoritmo tinklo kaštų dydžio prasme.

5) Modifikuotą skruzdžių kolonijos algoritmą fiksuoto telefono ryšio tinklo stočių išdėstymui. Algoritmas sukuriamas literatūroje [7] pateikiamą algoritmą modifikuojant šiais aspektais:

- Modifikuojama stoties patekimo į sprendinį tikimybės skaičiavimo metodika taip, kad tiktų fiksuoto telefono ryšio tinklo užduočiai spręsti;
- Sukuriama lokalios paieškos procedūra pagerinanti sprendinį tinklo kaštų dydžio prasme;

2. TEORINĖ DALIS

2.1. KOMBINATORINIO OPTIMIZAVIMO UŽDAVINIAI TELEKOMUNIKACIJOSE

Šioje darbo dalyje apžvelgsime kombinatorinio optimizavimo taikymą telekomunikacijų pramonėje. Pateiksime dažniausiai sprendžiamas problemas bei apžvelgsime kai kurias jų sprendimo galimybes kombinatorinio optimizavimo metodais.

2.1.1. KANALŲ PRISKYRIMO UŽDAVINYS

Kad tarp dviejų tinklo taškų būtų užmegztas ryšys, būtina jiems priskirti tam tikro dažnio kanalą. Kanalui dažnis yra parenkamas iš turimos dažnių juostos. Tinkle gali susidaryti tokia situacija, kad du ryšio kanalai gali įtakoti vienas kito duomenų perdavimo kokybę. Taip įvyks, jei:

- Abiejų kanalų signalų dažniai bus panašūs.
- Taškai, tarp kurių yra sudaryti ryšio perdavimo kanalai, yra arti vienas kito (geografiškai).

Sparčiai besivystant telekomunikacijų pramonei, valdžios institucijos pradėjo licencijuoti ryšio kanalų naudojimą bei už kiekvienos papildomos dažnių juostos naudojimą rinkti mokestį. Todėl bendrovėms atsirado poreikis optimaliai panaudoti turimus dažnius, t. y., verstis siauresne dažnių juosta, bet per daug nesumažinti ryšio kokybės. Šią problemą galime spręsti kombinatorinio optimizavimo metodais.

Literatūroje [2] ir [3] šią problemą siūloma spręsti šiais euristiniais metodais: „godaus“, lokalios paieškos, tabu paieškos ir kt. algoritmais. Dar, šio tipo uždavinius galima spręsti ir genetiniais, relaksaciniais bei kitais matematinio programavimo metodais.

Matematinė uždavinio formuluotė:

Duota:

$F = [f_{min}; f_{max}]$ - galimų naudoti dažnių juosta

N - tinklo stočių skaičius

$C = [c_{ij}]$ - kanalų suderinamumo matrica,

čia $i = \overline{1, N}$ ir $j = \overline{1, N}$

c_{ij} - minimalus atstumas tarp dažnių, priskiriamų stotims i ir j, kad nebūtų trukdžių

$M = (m_1, m_2, \ldots, m_N)$ m_i – sujungimų skaičius, reikalingas i-ajai stočiai

f_{ik}, $f_{ik} \in F$ - dažnis priskirtas k-tajam sujungimui i-ojoje stotyje

čia $i = \overline{1, N}$ ir $k = \overline{1, m_i}$

Rasti:

$$F^* = \{f_{ik} : \left|f_{ik} - f_{jl}\right| \geq c_{i,j}, \forall i, j, k, l \text{ išskyrus tada, kai } i = j \text{ ir } k = l\}, \text{ kad } \left|F^*\right| \to \min \qquad \textbf{(2.1)}$$

čia $i, j = \overline{1, N}$ ir $k, l = \overline{1, m_i}$

Suderinamumo matricą C galime interpretuoti kaip grafą G = (V, E), kur V – tinklo stočių aibė, o briaunų aibę E sudaro tokie lankai (i, j), kur $c_{ij} > 0$. Tuomet šitaip suformuluotą problemą laikysime grafo dažymo uždaviniu.

2.1.2. DUOMENŲ MARŠRUTIZAVIMO TINKLE UŽDAVINYS

Duomenys nuo vieno tinklo mazgo iki kito keliauja tinklu. Retai duomenų paketai iš vieno tinklo mazgo į kitą patenka tiesiogiai. Dažniausiai jie keliauja per kelis mazgus. Skirtingi paketai gali keliauti skirtingais keliais. Iškyla problema, kaip užprogramuoti tinklo mazgų adresų lenteles, kad duomenys pasiektų vieną ar kelis gavėjus nepažeisdami šių apribojimų:

- Duomenų paketai kelyje neužtruktų per ilgai („multimedia“ duomenų tipo atveju);
- Duomenų paketai pasiektų gavėjus keliaudami trumpiausią atstumą;

Formalizuosime šią užduotį:

Duota:

- Tinklo mazgų aibė;
- Kabelių aibė, kuriais mazgai sujungti į tinklą;
- Kiekvienam kabeliui žinoma:
  - Duomenų paketų perdavimo kaštai;
  - Maksimalus paketų skaičius, kuris gali būti perduotas vienu metu;
  - Duomenų paketų vėlinimas kelyje;

Uždučių variantai:

Perduoti duomenų paketus taip, kad:

- Perdavimo kaštai būtų kiek galima mažesni;
- Perdavimo metu kabelių pralaidumai išnaudojami kiek galima efektyviau;
- Paketų vėlinimas būtų ne didesnis už leistiną;

Sprendimo būdai:

1) Duomenų paketų perdavimo kaštų minimizavimas yra ekvivalentus minimalaus, padengiančio viršūnes, atitinkančias siuntėjus ir gavėjus, medžio grafe paieškai. Literatūroje [5] pateikiama nemažai algoritmų šiai užduočiai atlikti. Paminėsime, kad dažniausiai tai atliekama kruskal‘s ir prim‘s algoritmais.

2) Tinklo perkrovų minimizavimas atliekamas minimizuojant maksimalų kabelio apkrovos ir jo pralaidumo skirtumą. Šio tipo uždaviniai formaliai vadinami „multicast packing problems“. Literatūroje [5] pateikiami tokiomis idėjomis besiremiantys sprendimo variantai: minimalaus padengiančio norimas viršūnes medžio paieška, kirtimų aibės paieška.

3) Tarkime, kad norime atlikti duomenų paketų vėlinimo minimizavimą. Tuomet perdavimo kabeliu vėlinimui suteikiame atstumo tarp dviejų mazgų interpretaciją ir minėtą uždavinį keičiame į trumpiausio kelio paieškos grafe uždavinį, kurį, kaip nurodoma literatūroje [5] galima spręsti dikstros algoritmu ir įvairiomis jo modifikacijomis.

4) Sudėtingesniais atvejais paketo perdavimo kabeliu kaina ar trukmė gali būti atsitiktinis dydis. Tuomet, kaip nurodoma literatūroje [7], trumpiausio kelio paieškos uždavinys gali būti sprendžiamas „Skruzdžių kolonijos“ algoritmu.

Realiuose praktiniuose uždaviniuose aprašytieji uždaviniai yra sujungiami. Pvz., stengiamasi minimizuoti duomenų perdavimo kaštus įvedus vėlinimo arba tinklo apkrovos apribojimus.

Matematinė uždavinio formuluotė:

Duota:

$G = (V; E)$ V – tinklo mazgų aibė; E – aibė briaunų, reprezentuojančių visus įmanomus sujungimus tarp tinklo taškų

$c(i, j) \geq 0,$ - c(i, j) kabelio tarp mazgų i ir j pralaidumas

$\forall (i, j) \in E : w(i, j) \geq 0,$ - w(i, j) duomenų perdavimo tarp mazgų i ir j kaštai

$d(i, j) \geq 0$ - d(i, j) duomenų paketų vėlinimas kelyje tarp mazgų i ir j

Rasti:

$P = \{v_{i_1}, ..., v_{i_j}\},$
čia $(v_{i_k}, v_{i_{k+1}}) \in E, \forall k \in \{1, ..., j-1\}$ - kelias tarp siuntėjo ir gavėjo

$$1)\ w(P) \to \min, w(P) := \sum_{k=1}^{j-1} w(i_k, i_{k+1}) \qquad \textbf{(2.2)}$$

- sąlyga, kad kaštų suma perduodant duomenis būtų minimali

2) $p(V) \to \min, \mathrm{p}(V) := \max_{(i,j)\in E}\{c(i,j) - u(i,j)\}$, u(i, j) – kabelio tarp mazgų i ir j apkrovimas **(2.3)**

- sąlyga, kad kabelių pralaidumas būtų išnaudojamas kiek įmanoma efektyviau

3) $d(P) \to \min, \mathrm{d}(P) := \sum_{k=1}^{j-1} d(i_k, i_{k+1})$ **(2.4)**

- sąlyga, kad paketų vėlinimas būtų minimalus

2.1.3. FIKSUOTO TELEFONO RYŠIO TINKLO OPTIMIZAVIMO UŽDAVINYS

Tarkime, kad turime aibę mazgų, kuriuos norime sujungti į vieną tinklą, taip kad tarp bet kurių tinklo mazgų būtų galima užmegzti ryšį. Tinklą reikia sukonstruoti taip, kad išlaidos būtų kliek galima mažesnės. Formalizuosime šią užduotį:

Duota:

- Viršūnių aibė (butai, gyvenamieji namai, įstaigos, tarpinės stotys, centrinės stotys ir pan.), kurias reikia sujungti į tinklą;
- Briaunų aibė (kabeliai), visi įmanomi mazgų sujungimo variantai;
- Kiekvienai briaunai priskirtas neneigiamas skaičius – kabelio instaliavimo kaina;

Užduotis:

- Išrinkti tas briaunas, kurių instaliavimo kainų suma minimali;
- Sujungus viršūnes išrinktomis briaunomis ryšys įmanomas tarp bet kurių viršūnių.

Taip suformuluoto uždavinio sprendimas keičiamas į minimalaus, padengiančio visas viršūnes, medžio paiešką. Literatūroje [4] nurodoma, kad tai galima atlikti šiais algoritmais: kruskal‘s, prim‘s ir kt.

Matematinė uždavinio formuluotė:

Duota:

$G = (V;E)$	V – aibė taškų, kuriuos reikia sujungti; E – aibė briaunų, reprezentuojančių visus įmanomus sujungimus tarp tinklo taškų
$\forall e \in E : c_e \geq 0$	c_e- kabelio instaliavimo kaina, priskirta kiekvienai briaunai

Rasti:

$T = (V;E_{min})$ V – aibė taškų, kuriuos reikia sujungti;

E_{min} – aibė tokių briaunų, kad $\forall v, w \in V$ yra sujungti keliu;

$$\forall E' \subseteq E : c(E_{\min}) \le c(E') \text{, čia } c(E) := \sum_{e \in E} c_e \qquad \textbf{(2.5)}$$

2.1.4. MOBILIOJO TELEFONO RYŠIO TINKLO OPTIMIZAVIMO UŽDAVINYS

Spręsime šį uždavinį vadovaudamiesi šiomis prielaidomis:

- Laikysime, kad ryšio poreikis yra tolygus visoje teritorijoje.
- Tinklo padengimo kaina lygi visų siųstuvų teritorijoje kainų sumai.
- Pradiniai duomenys yra potencialios siųstuvų buvimo vietos ir jiems priskirtos kainos.
- Kiekvienam siųstuvui yra priskiriama teritorija, kurią jis padengia.

Algoritmo tikslas, išrinkti iš visos vartotojo įvestos siųstuvų aibės tokį poaibį, kad šis padengtų nemažiau už nustatytą procentą teritorijos ir to poaibio siųstuvų kaštų suma būtų patenkinama. Taip apibrėžtas uždavinys galėtų būti sprendžiamas šiais grafų teorija paremtais algoritmais:

1) Maksimalios nepriklausomos aibės paieškos algoritmas.

Matematinė uždavinio formuluotė:

Duota:

$G = (V;E)$ V – vartotojo įvesta stočių aibė

$$\forall v, w \in V \text{ lankas } (v, w) \in E, \text{ jei } \frac{S(v) \cap S(w)}{\min\{S(v), S(w)\}} \ge K, \qquad \textbf{(2.6)}$$

čia K algoritmo parametras, o S(x) – stoties x padengiamos teritorijos plotas

Rasti:

$$V' : V' \cap \Gamma(V') = \varnothing \text{ ir } |V'| \to \max \qquad \textbf{(2.7)}$$

V' – maksimali nepriklausoma grafo G aibė

2) Minimalios dominuojančios aibės algoritmas.

Matematinė uždavinio formuluotė:

Duota:

$G = (V;E)$ $V = A \cup B$, čia A – visų galimų stočių dislokacijos vietų aibė

B – visų padengiamų vietų aibė

$\forall v, w \in V$ lankas $(v, w) \in E$, jei $v \in A$, o w ∈ B, t. y., stotis v padengia vietovę w

Rasti:

$$V' : V' \cup \Gamma(V') = V \text{ ir } |V'| \to \min \qquad \textbf{(2.8)}$$

V' – minimali dominuojanti grafo G aibė.

Kaip nurodoma literatūroje [1], realizuojant šiuos modelius kompiuteriniu variantu, maksimalios nepriklausomos aibės paieška atliekama „godžiu“ algoritmu, kai tuo tarpu minimali dominuojanti aibė gali būti skaičiuojama genetiniais algoritmais.

3) „Skruzdžių kolonijos“ algoritmas.

Matematinė uždavinio formuluotė:

Duota:

$A = (a_{ij})$ A – $m \times n$ matavimų matrica, kur $a_{ij} = \begin{cases} 1, \ j-asis\ stulpelis\ padengia\ i-ąją\ eilutę \\ 0, \ kitais\ atvejais \end{cases}$

$\forall j := b_j, b_j \geq 0$ b_j – kiekvienam stulpeliui, priskirti stoties pastatymo kaštai

Rasti:

$$f(y) = \sum_{j=1}^{n} b_j \cdot y_j \to \min \qquad \textbf{(2.9)}$$

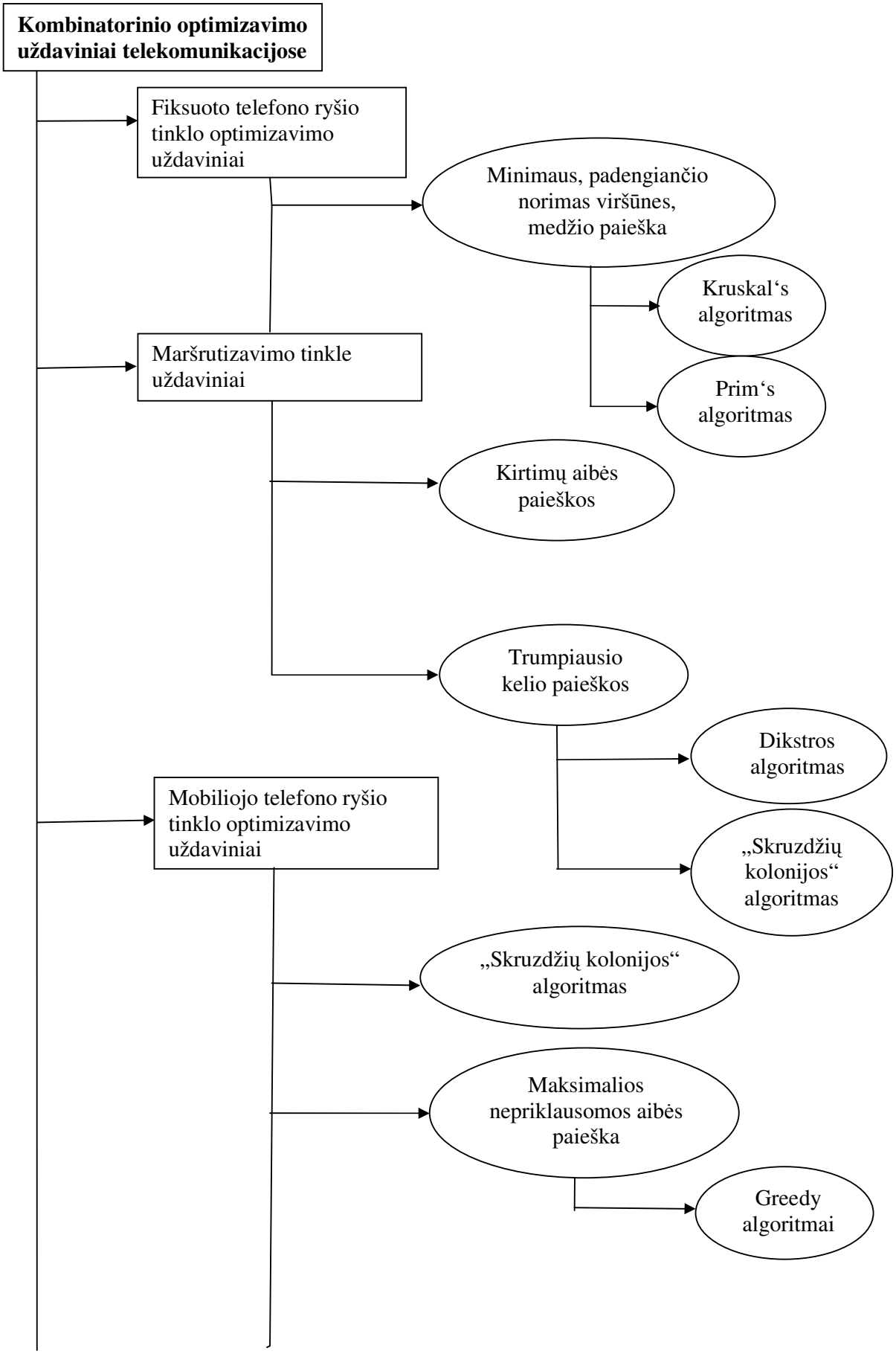
$$\sum_{j=1}^{n} a_{ij} \cdot y_j \geq 1, \ i = \overline{1, m} \qquad y_j = \begin{cases} 1, \ jei\ j-oji\ kolona\ priklauso\ sprendiniui \\ 0, \ kitais\ atvejais \end{cases} \qquad \textbf{(2.10)}$$

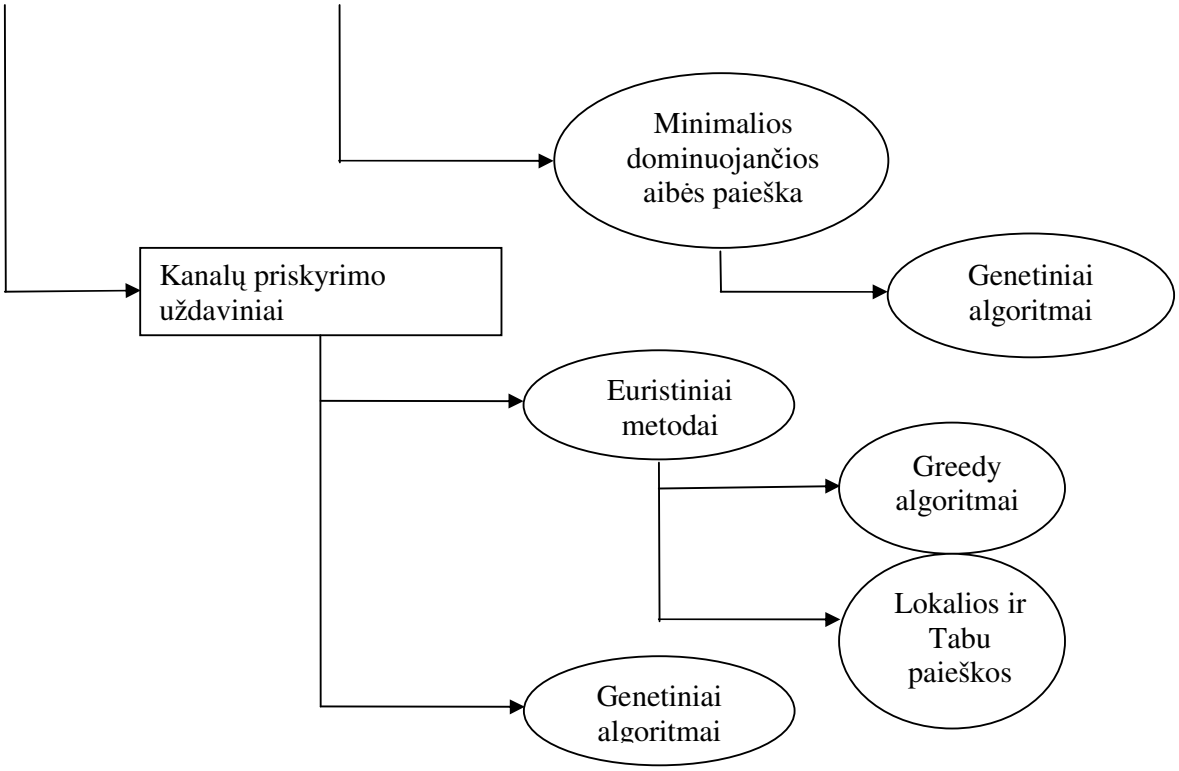
$y_j \in \{0, 1\}, \ j = \overline{1, n},$

Atsižvelgę į padarytas prielaidas galime teigti, kad šie modeliai sprendžia ganėtinai abstrakčias teritorijos padengimo užduotis. Norint šiuos modelius taikyti realybėje, būtina įvertinti daugelį kitų parametrų. Pavyzdžiui, vartotojai gali būti pasiskirstę netolygiai visoje teritorijoje, o koncentruotis tam tikrose jos vietose (miestuose). Išrenkant siųstuvų dislokacijos vietas būtina įvertini ir daugybę siųstuvo charakteristikų bei parametrų, ne vien tik teritorijos, kurią jis gali aptarnauti, plotą.

2.2. UŽDAVINIŲ IR ALGORITMŲ KLASIFIKACIJA

2.1. Skyrelyje aptartas problemas bei jų sprendimo metodų klasifikaciją pateikiame 2.1 pav.:


Kombinatorinio optimizavimo uždaviniai telekomunikacijose
Fiksuoto telefono ryšio tinklo optimizavimo uždaviniai
Minimaus, padengiančio norimas viršūnes, medžio paieška
Kruskal‘s algoritmas
Prim‘s algoritmas
Maršrutizavimo tinkle uždaviniai
Kirtimų aibės paieškos
Trumpiausio kelio paieškos
Dikstros algoritmas
„Skruzdžių kolonijos“ algoritmas
Mobiliojo telefono ryšio tinklo optimizavimo uždaviniai
„Skruzdžių kolonijos“ algoritmas
Maksimalios nepriklausomos aibės paieška
Greedy algoritmai



2.1 pav. Kombinatorinio optimizavimo uždavinių ir metodų schema

2.3. DAUGIAPARAMETRINIŲ OPTIMIZAVIMO METODŲ APŽVALGA

Daugiaparametrinio optimizavimo atveju ieškant optimalaus sprendinio reikia nagrinėti daugelį parametrų, nuo kurių priklauso optimizuojamo objekto reikšmė. Dažniausiai (ypač realaus pasaulio uždaviniuose) optimalaus sprendinio paieškos metu tenka kontroliuoti faktorius, kurie taip pat priklauso nuo optimizavimo parametrų. Tai gali būti objekto kokybė, jo patikimumas ar kita. Šie faktoriai apriboja optimalaus sprendinio paieškos sritį ir yra vadinami apribojimais.

Sprendžiant realaus pasaulio uždavinius dažniausiai susiduriama su įvairių rūšių ir tipų priklausomybėmis tarp kintamųjų ir kokybės rodiklių arba tarp kintamųjų ir tikslo funkcijos. Visa tai vadinama netiesinėmis priklausomybėmis ir suvedama į netiesinio programavimo uždavinį. Vienas iš apribojimų pažeidimų įvertinimo metodų tai literatūroje [6] pateikiamas baudos funkcijų metodas. Pagal šį metodą prie minimizuojamos tikslo funkcijos pridedamos baudos, kurios didėja atsiradus naujiems pažeidimams.

$$F = C_{zj} \cdot \overline{Z}_j + \sum_{i=1}^{n} (C_{yi} \cdot \Delta \overline{Y}_i^2) + \sum_{k=1}^{m} (C_{xk} \cdot \Delta \overline{X}_k^2); \qquad \textbf{(2.11)}$$

čia C_{zj} – įtakos koeficientas optimizavimo parametrui Z_j;

C_{yi} – baudos koeficientas pridedamas prie tikslo funkcijos įvykus apribojimų Y_i pažeidimui;

C_{xk} – baudos koeficientas pridedamas prie tikslo funkcijos, kai nepriklausomi kintamieji X_k nepatenka į leistinų reikšmių sritį

$$\overline{Z}_j = Z_j / Z_{j\_mean} \quad \textbf{(2.12)}$$

– optimizuojamo parametro Z_j, vidutinė reikšmė;

$$\Delta\overline{Y}_i = \begin{cases} \dfrac{Y_i - Y_{i\min}}{Y_{imean}}, & \text{kai} \quad Y_i < Y_{i\min} \\ 0, & \text{kai} \quad Y_{i\min} \le Y_i \le Y_{i\max} \\ \dfrac{Y_i - Y_{i\max}}{Y_{imean}}, & \text{kai} \quad Y_i > Y_{i\max} \end{cases} \quad \textbf{(2.13)}$$

– santykinis apribojimų Y_i pažeidimo dydis;

$$\Delta\overline{X}_k = \begin{cases} \dfrac{X_k - X_{k\min}}{X_{kmean}}, & \text{kai} \quad X_k < X_{k\min} \\ 0, & \text{kai} \quad X_{k\min} \le X_k \le X_{k\max} \\ \dfrac{X_k - X_{k\max}}{X_{kmean}}, & \text{kai} \quad X_k > X_{k\max} \end{cases} \quad \textbf{(2.14)}$$

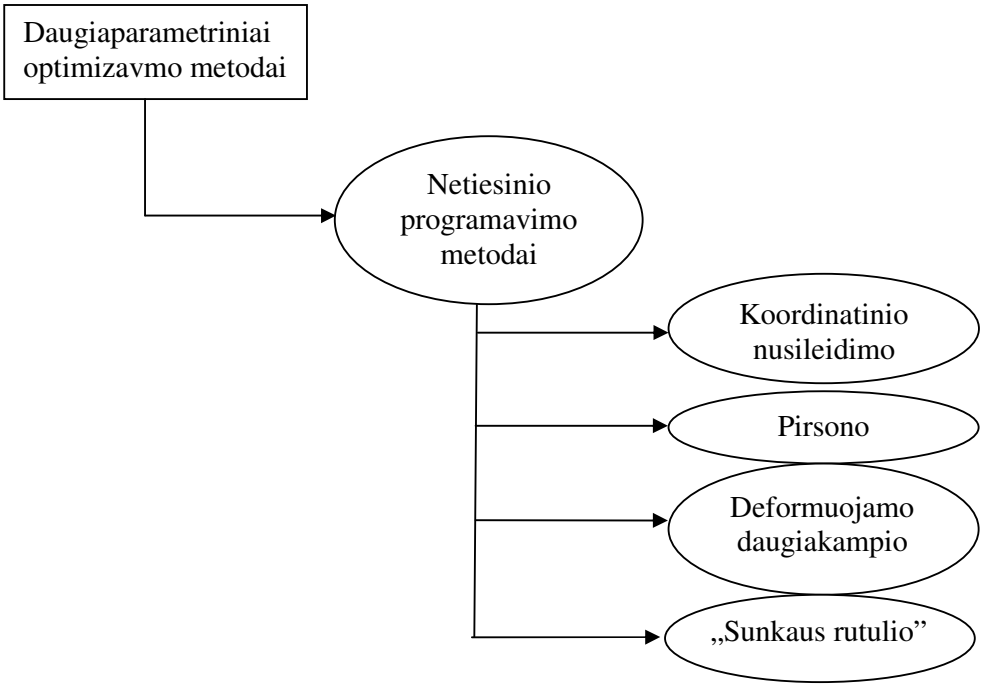
– santykinis nepriklausomų kintamųjų X_k išėjimo iš leistinų reikšmių srities dydis

Kaip nurodoma literatūroje [6], šio uždavinio sprendimui gali būti naudojami įvairūs netiesinio programavimo metodai: koordinatinio nusileidimo, „sunkaus rutulio“, Pirsono, deformuojamo daugiakampio ir kiti metodai. Tačiau visi šie minimi metodai turi vieną bendrą trūkumą: jie gali būti naudojami tik tada, kai tiek tikslo, tiek apribojimų funkcijos aprašomos tolydžiomis diferencijuojamomis funkcijomis. Prieigos tinklo optimizavimo atveju dauguma kintamųjų reikšmių yra diskrečios, o kai kurių netgi neskaitinės, pavyzdžiui, moduliacijos tipų: QPSK, 4QAM, 16QAM ir kitų, dėl to programavimo metodai, paremti operacijomis, kurioms apibrėžti reikalingos tolydžios priklausomybės, šiame darbe nagrinėjamo uždavinio sprendimui negali būti naudojami. Nors aprėpčių zonų dydžiai iš prigimties yra tolydūs, bet tradiciškai yra naudojamos tik diskrečios šių kintamųjų reikšmės. Be to, pasirinkus diskrečias kintamųjų reikšmes, nereikia naudoti sudėtingų mišraus programavimo metodų. Taigi, gauname diskretaus optimizavimo uždavinį.

Kadangi sprendžiant prieigos tinklo optimizavimo uždavinį bus naudojamas plataus spektro apribojimų rinkinys, o parametrai - pakankamai skirtingo pobūdžio, tai tiesiogiai taikyti optimizavimo teorijoje naudojamus metodus yra komplikuota. Todėl šiame darbe nagrinėjamą optimizavimo uždavinį spręsime pasitelkdami matematinio ir telekomunikacijų tinklų optimizavimo metodikų elementus:

- daugiaparametrinį optimizavimą;

- parametrų apibrėžimo srities apribojimus;
- kintamųjų ir jų reikšmių surikiavimą pagal įtaką tikslo ir apribojimų funkcijoms;
- minimalių kaštų ir tinklo kokybės charakteristikų įvertinimo metodikas;
- meta-euristinius optimizavimo algoritmus.



2.2 pav. Daugiaparametrinių optimizavimo metodų schema

2.4. UŽDAVINIO FORMULAVIMAS

Šiame darbe nagrinėjamo uždavinio tikslo funkcija, tai prieigos tinklo kaštų minimizavimo funkcija, išreiškiama tokia priklausomybe:

$$C_{min}=f(L, T), \text{ čia } T \in \mathbf{T}, \quad \text{kai } T=(M,R,H,V,W,G). \tag{2.15}$$

Šioje išraiškoje C – tinklo įdiegimo kaštai;

L – stoties aprėpties zonos dydis (spindulys), km;

T – tinklo prieigoje naudojamos technologijos kintamieji:

M – kanalo/perdavimo terpės tipas,

R – duomenų perdavimo sparta, kbps,

H – duomenų paketų aptarnavimo disciplina,

V – prieigos prie kanalo metodas,

W – moduliacijos tipas,

G – kodeko tipas.

Išraiškoje (2.15) kintamųjų aibė T aprašoma taip:

$$\mathrm{T} = \mathrm{M} \times \mathrm{R} \times \mathrm{H} \times \mathrm{V} \times \mathrm{W} \times \mathrm{G}, \qquad \textbf{(2.16)}$$

čia

$$\mathrm{L} = \{L_1, L_2, L_3, ..., L_{n_2}\},\ pavyzdžiui, \quad \mathrm{L} = \{1,2,3,...,15\},$$

$$\mathrm{M} = \{M_1, M_2, M_3, ..., M_{n_3}\},\ pavyzdžiui, \quad \mathrm{M} = \{optika, radijo, laidinis\},$$

$$\mathrm{R} = \{R_1, R_2, R_3, ..., R_{n_4}\},\ pavyzdžiui, \quad \mathrm{R} = \{64,128,192,...,34000\},$$

$$\mathrm{H} = \{H_1, H_2, H_3, ..., H_{n_5}\},\ pavyzdžiui, \quad \mathrm{H} = \{M/M/1, M/M/1/S, M/D/1, M/G/1, ...\},$$

$$\mathrm{V} = \{V_1, V_2, V_3, ..., V_{n_6}\},\ pavyzdžiui, \quad \mathrm{V} = \{FDMA, TDMA, CDMA, CSMA/CD, ...\},$$

$$\mathrm{W} = \{W_1, W_2, W_3, ..., W_{n_7}\},\ pavyzdžiui, \quad \mathrm{W} = \{QPSK, 16QAM, 32QAM, 64QAM, ...\},$$

$$\mathrm{G} = \{G_1, G_2, G_3, ..., G_{n_8}\},\ pavyzdžiui, \quad \mathrm{G} = \{G.711, G.722, G.723, G.726, G.728, G.729, ...\}.$$

čia n_1, n_2,.., n_8– kintamųjų reikšmių skaičius.

Suformuluosime optimizavimo užduotį: tarkim, duota funkcija *C=f (L, T)* ir sąlygos, kurias tenkina nežinomieji *(L, T),* reikia rasti kintamųjų reikšmes (*L*, T*),* kurioms esant funkcijos reikšmė būtų minimali, t.y., $C_{min}=f(L^*, T^*)$. *N* – matį tašką $C_{ls}=f(L, T)$, tenkinantį apribojimų sistemą:

$$\begin{cases} Q_{1\min} \le Q_1 \le Q_{1\max}, \\ Q_{2\min} \le Q_2 \le Q_{2\max}, \\ Q_{3\min} \le Q_3 \le Q_{3\min}, \\ ......, \\ Q_{r\min} \le Q_r \le Q_{r\max}, \end{cases} \qquad \textbf{(2.17)}$$

vadinsime leistinuoju sprendiniu. Čia Q_1, Q_2,..Q_r – *r*–asis iš pradinės prieigos tinklo struktūros nustatytas kokybės rodiklis; Q_{1min}, Q_{2min},..Q_{rmin}, Q_{1max}, Q_{2max},..Q_{rmax} – *r* –ojo užduoto kokybės rodiklio minimali/maksimali reikšmė.

Kadangi kokybės rodikliai gali būti apriboti arba iš apačios, arba iš viršaus reikšmėmis Q_{imin}, Q_{imax}, be to, jų reikšmės gali sutapti su kokybės normų ribinėmis reikšmėmis, todėl apribojimai (2.17) įvedami negriežtomis nelygybėmis.

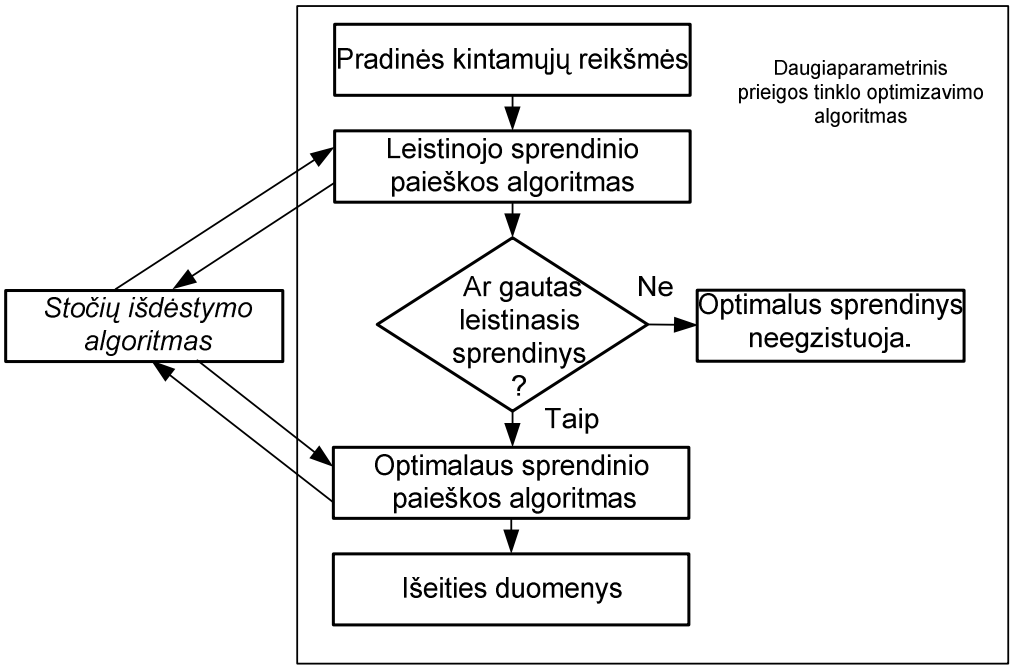
Jeigu leistinųjų sprendinių aibė nėra tuščia, tuomet pereiname prie optimalaus sprendinio paieškos. Funkcijos *C=f(L, T)* optimaliu sprendiniu vadinsime tašką, kuriame kintamųjų reikšmės patenka į leistinųjų sprendinių aibę ir funkcijos *C* reikšmė yra minimali:

$$C_{min}=f(L^*, T^*)=min. \qquad \textbf{(2.18)}$$

2.5. PRIEIGOS TINKLO OPTIMIZAVIMO ALGORITMAS

Šiame skyrelyje pateiksime algoritmo, skirto 2.4 skyrelyje suformuluotam uždaviniui spręsti principus.

Nubraižome optimizavimo etapų blokinę schemą:



2.3 pav. Optimizavimo etapų blokinė schema.

Optimizavimo algoritmą sudaro šie žingsniai:

1 punktas

Pasirenkami optimizavimo kintamieji bei kokybės rodikliai. Užduodamos reikšmės, kurias turi tenkinti kokybės rodikliai. Įvedamos galimos kintamųjų reikšmės. Užduodamos kintamųjų bei jų reikšmių rikiuotės pagal tinklo kokybės parametrus ir tinklo kaštus.

2 punktas

Fiksuojame pradinį parametrų reikšmių rinkinį. Tai galime padaryti įvairiais būdais: Atsitiktinai pasirenkant reikšmes, imant pirmą reikšmę iš kintamųjų sąrašo arba pateikiant įvesti vartotojui. Užfiksavę reikšmes pereiname prie leistinojo sprendinio paieškos.

Šioje dalyje iš pradinės prieigos tinklo struktūros nustatyti kokybės rodikliai Q_1, Q_2,..Q_r patikrinami, ar tenkina užduotas kokybės normas, pagal (2.17) sąlygas. Jeigu bent vienas rodiklis netenkina sąlygos, tuomet keičiama kintamojo, kuris labiausiai įtakoja neatitikusį kokybės normų

rodiklį, reikšmė. Vykdomas stočių išdėstymo algoritmas. Procesas tęsiamas tol, kol randamas leistinas sprendinys arba perrenkamos visų kintamųjų, įtakojančių pažeistą apribojimą, reikšmės.

3 punktas

Jeigu neradome tokio reikšmių rinkinio, kad būtų tenkinamos visos kokybės normos, darbą baigiame – leistinas sprendinys neegzistuoja, juo labiau neegzistuoja ir optimalus sprendinys. Kitu atveju, jeigu radome tokį parametrų reikšmių rinkinį, kad tenkinamos (2.17) sąlygos, pereiname prie optimalaus sprendinio paieškos.

4 punktas

Ieškant optimalaus sprendinio, keičiama kintamojo, kuris labiausiai įtakoja tikslo funkciją, reikšmė taip, kad tikslo funkcija mažėtų. Vykdomas stočių išdėstymo algoritmas. Jei išdėsčius stotis neatsiranda apribojimų pažeidimų, imama sekanti reikšmė ir t. t. Kai nebegalima daugiau keisti, einame prie kito kintamojo. Procesas tęsiamas tol, kol peržiūrimi visi kintamieji, turintys įtakos tikslo funkcijos reikšmei.

5 punktas

Grąžinami išeities duomenys. Išeities duomenis sudaro kintamųjų reikšmių rinkinys, tenkinantis užduotas kokybės normas ir minimizuojantis tikslo funkciją, bei tikslo funkcijos reikšmė tame taške.

3. TIRIAMOJI DALIS IR REZULTATAI

Šiame skyriuje spręsime 2.4 skyrelyje suformuluotą prieigos tinklo optimizavimo uždavinį. Mūsų tikslas taip parinkti tinklo parametrus, kad būtų tenkinami reikalavimai tinklo kokybei, o tinklo kaina minimali. Naudodami stočių išdėstymo algoritmus apskaičiuosime optimalias stočių dislokacijos koordinates. Kitame skyrelyje pateikiame algoritmo, skirto šiam uždaviniui spręsti, realizaciją.

3.1. LEISTINO SPRENDINIO PAIEŠKOS ALGORITMAS

1 punktas

Kintamieji ir jų reikšmės sutvarkomos pagal įtaką kaštams ir kokybės kriterijams:

$$\left(X_1^{k,l_1}, X_2^{k,l_2}, \cdots, X_n^{k,l_n}\right), \qquad \textbf{(3.1)}$$

čia k – kintamųjų X įtakos tvarkos indeksas, $k = 0$ – kaštams, $k = \overline{1,r}$ – kokybės kriterijui Q_k, r – kokybės kriterijų (apribojimų) skaičius, $l_s \in \{1,2,\cdots,n\}$ – kintamojo vieta pradiniame kintamųjų sąraše, n – kintamųjų skaičius.

Kintamojo $X_i^{k,l}$ reikšmių sutvarkyta aibė: $\left\{x_{i,1}^{k,l_i,t_1}, x_{i,2}^{k,l_i,t_2}, \cdots, x_{i,m_i}^{k,l_i,t_{m_i}}\right\}$, **(3.2)**

čia $t_j \in \{1,2,\cdots,m_i\}$ kintamojo $X_i^{k,l}$ j–osios reikšmės vieta pradiniame kintamojo reikšmių sąraše.

2 punktas

Pasirenkamas pradinis kintamųjų reikšmių rinkinys $(x_1, x_2, \cdots, x_n)$ (kintamųjų ir jų reikšmių indeksai pagal pradinius sąrašus).

3 punktas

Pagal pasirinktas kintamųjų reikšmes apskaičiuojami kokybės rodikliai Q_1, Q_2,..Q_r bei išdėstomos stotys pagal 3.3 skyriuje aprašytas metodikas.

4 punktas

Randamas apribojimų pažeidimų vektorius $P = (p_1, p_2, \cdots, p_r)$, kuris apibrėžiamas tokia formule:

$$p_k = \begin{cases} 0, \text{ jei } Q_{k\,\min} \le Q_k \le Q_{k\,\max} \\ -1, \text{ jei } Q_k < Q_{k\,\min} \\ 1, \text{ jei } Q_{k\,\max} < Q_k \end{cases}, \; k = \overline{1,r}\,. \qquad \textbf{(3.3)}$$

Jei $P = (0,\cdots,0)$, rastas leistinasis sprendinys ir pereinama į optimalaus sprendinio paiešką.

Vektorių P, kurio visos elementų reikšmės lygios nuliui, žymėsime P_0 ir vadinsime nuliniu $P_0 = (0,\cdots,0)$.

5 punktas

Eilės tvarka nagrinėjame apribojimą Q_{k_p}, kuriam atitinkama komponentė $p_{k_p} \neq 0$ pažeidimų vektoriuje P.

Jei P = P_0 – visi pažeidimai ištaisyti sėkmingai, baigiame darbą ir grąžiname pranešimą, kad rastas leistinas sprendinys.

6 punktas

Eilės tvarka, pagal įtaką apribojimui Q_{k_p}, parenkamas kintamasis ir nagrinėjamos jo reikšmės. Jei nebėra dar neperžiūrėtų kintamųjų, ištaisyti k_p-ojo pažeidimo nepavyko. Baigiame darbą ir grąžiname pranešimą, kad leistinas sprendinys neegzistuoja.

7 punktas

Priklausomai nuo p_{k_p} reikšmės, kuri bus arba $p_{k_p} = -1$, arba $p_{k_p} = 1$, nagrinėjamo kintamojo reikšmė parenkama pagal šias formules:

$$\begin{cases} jei \quad p_{k_p} = -1, tai \quad Q_{k_p}\left(x_{i,j}^{k_p,l_i,t_j}\right) < Q_{k_p}\left(x_{i,j+1}^{k_p,l_i,t_{j+1}}\right) \\ jei \quad p_{k_p} = 1, tai \quad Q_{k_p}\left(x_{i,j}^{k_p,l_i,t_j}\right) > Q_{k_p}\left(x_{i,j-1}^{k_p,l_i,t_{j-1}}\right) \end{cases} \qquad \textbf{(3.4)}$$

Atliekami 3 ir 4 punktai.

7 punktas kartojamas tol, kol tenkinamos šios keturios sąlygos:

- $p_{k_p} \neq 0$
- $j > 0, jei\ p_{k_p} = -1$
- $j < n, jei\ p_{k_p} = 1$ **(3.5)**
- Atlikus reikšmės pakeitimą pažeidimų vektoriuje P neatsirado naujų pažeidimų. T. y., pažeidimų vektoriaus elementai vietoj nulinės reikšmės neįgijo -1, 1 ir nepasikeitė pažeidimo tipas: vietoj -1 neatsirado 1 arba vietoj 1 neatsirado -1. Ši sąlyga bus išpildoma, jei galios šie sąryšiai:

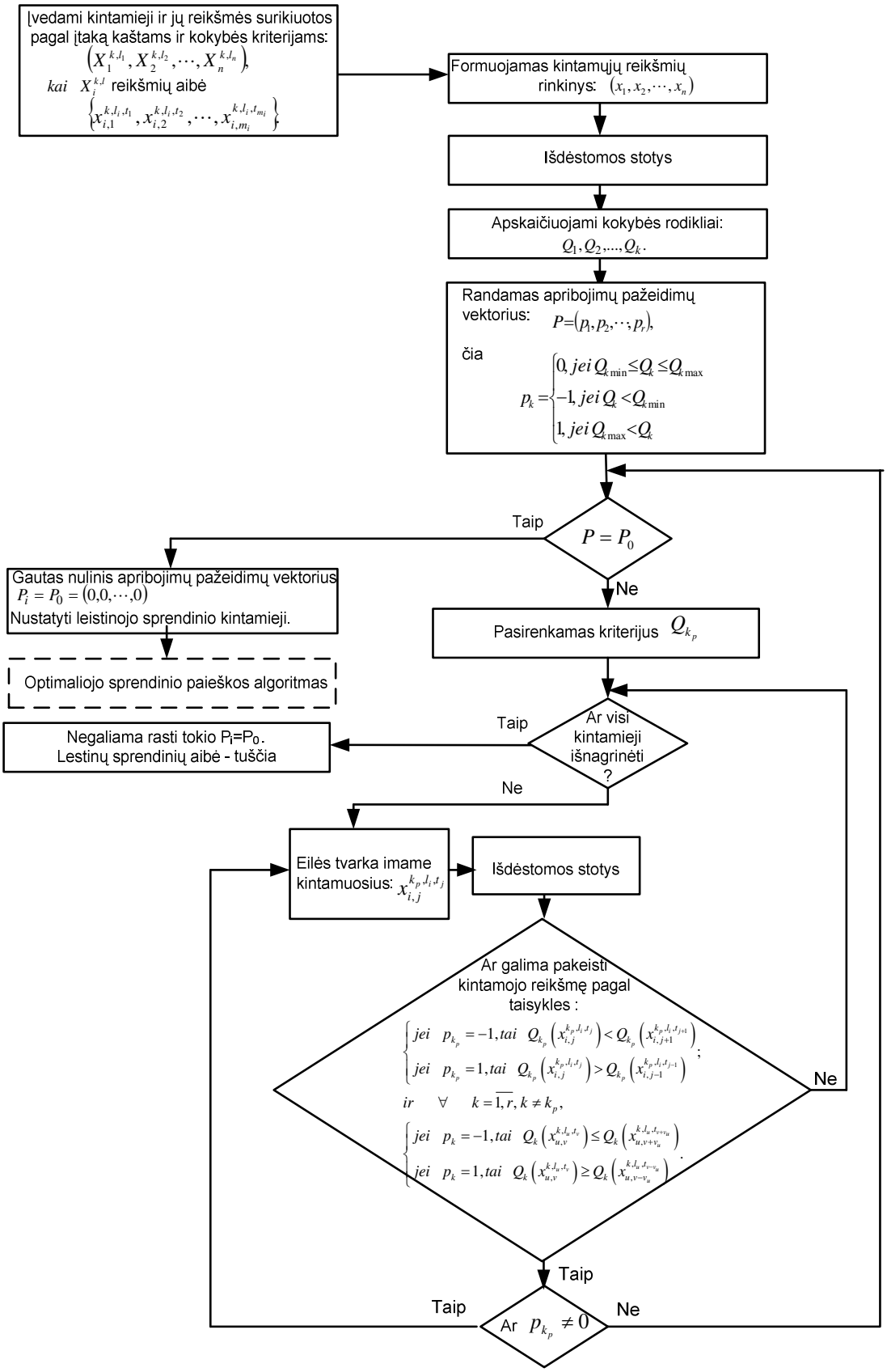
$$\begin{cases} jei \quad p_k = -1, tai \quad Q_k\left(x_{u,v}^{k,l_u,t_v}\right) \leq Q_k\left(x_{u,v+v_u}^{k,l_u,t_{v+v_u}}\right) \\ jei \quad p_k = 1, tai \quad Q_k\left(x_{u,v}^{k,l_u,t_v}\right) \geq Q_k\left(x_{u,v-v_u}^{k,l_u,t_{v-v_u}}\right) \end{cases}, \ \forall k = \overline{1,r} \text{ ir } \ k \neq k_p, \qquad \textbf{(3.6)}$$

čia $l_i = l_u$, $t_j = t_v$, $t_{j+1} = t_{v+v_u}$ arba $t_{j+1} = t_{v-v_u}$, arba $t_{j-1} = t_{v+v_u}$, arba $t_{j-1} = t_{v-v_u}$, $v_u \geq 1$.

8 punktas

Jei pažeista antra, trečia arba ketvirta 7 punkto vykdymo sąlyga, kartojame 6 punktą. Jei pažeista pirmoji sąlyga, tuomet k_p-asis pažeidimas ištaisytas – kartojame 5 punktą.

Nubraižome leistinojo sprendinio paieškos blokinę schemą:



3.1 pav. Leistinojo sprendinio paieškos blokinė schema

2.2. OPTIMALAUS SPRENDINIO PAIEŠKOS ALGORITMAS

1 punktas

Pasirinktam kintamajam eilės tvarka imama tikslo funkcijos reikšmę *C* mažinanti reikšmė. Tuomet galios sąryšis:

$$C\left(x_{i,j}^{0,l_i,t_j}\right) > C\left(x_{i,j-1}^{0,l_i,t_{j-1}}\right). \qquad \textbf{(3.7)}$$

2 punktas

Pagal pasirinktas kintamųjų reikšmes apskaičiuojami kokybės rodikliai Q_1, Q_2,..Q_r. Išdėstomos stotys pagal 3.3 skyriuje aprašytas metodikas.

3 punktas

Tikrinama, ar toks kintamojo reikšmės pakeitimas nepažeis apribojimų, ar galios nelygybės:

$$Q_{k\,\min} \le Q_k\left(x_{u,v}^{k,l_u,t_v}\right) \le Q_{k\,\max},\ k=\overline{1,r},\ l_i = l_u,\ t_{j-1} = t_v. \qquad \textbf{(3.8)}$$

4 punktas

Jei apribojimai nepažeidžiami, keičiama pasirinkto kintamojo reikšmė (indeksas $j = j-1$) ir kartojamas 2 punktas.

5 punktas

Jei apribojimai pažeidžiami, atliekamas pažeidimų taisymo algoritmas (3.2.1 skyrelis)

6 punktas

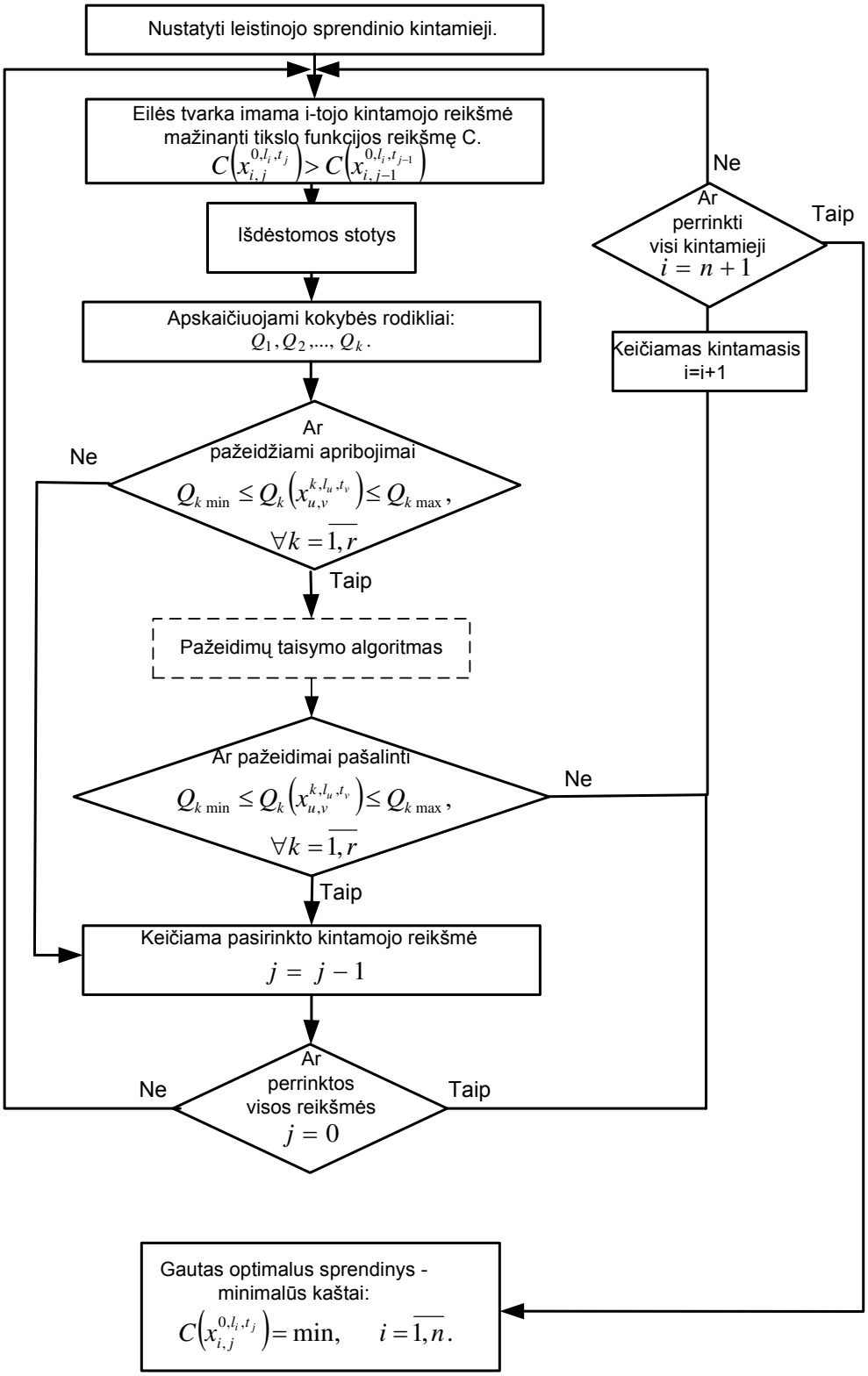
Jei apribojimų pažeidimai sėkmingai ištaisomi, galioja (2.17) nelygybės, tai keičiama pasirinkto kintamojo reikšmė (indeksas $j = j-1$) ir kartojamas 1 punktas.

Jei ne - kartojamas 1 punktas (indeksas $i = i+1$).

7 punktas

Kai nebeįmanoma pakeisti nei vieno kintamojo reikšmės, nuo kurių priklauso kaštų funkcija, gaunamas optimalus sprendinys.

Nubraižome optimaliojo sprendinio suradimo algoritmo blokinę schemą:



3.2 pav. Optimalaus sprendinio suradimo algoritmas

3.2.1. PAŽEIDIMŲ TAISYMO ALGORITMAS

1 punktas

Algoritmui paduodamas pradinis pažeidimų vektorius P.

2 punktas

Eilės tvarka nagrinėjame apribojimą Q_{k_p}, kuriam atitinkama komponentė $p_{k_p} \neq 0$ pažeidimų vektoriuje P. Jei P = P_0, visi pažeidimai ištaisyti sėkmingai, baigiame darbą ir grąžiname pranešimą, kad pažeidimai ištaisyti.

3 punktas

Eilės tvarka, pagal įtaką apribojimui Q_{k_p} parenkamas kintamasis, nuo kurio nepriklauso kaštų funkcijos reikšmė ir nagrinėjamos jo reikšmės.

Jei nebėra dar neperžiūrėtų kintamųjų, ištaisyti k_p-tojo pažeidimo nepavyko. Baigiame darbą ir grąžiname pranešimą, kad nepavyko ištaisyti pažeidimų.

4 punktas

Priklausomai nuo p_{k_p} reikšmės, kuri bus arba $p_{k_p} = -1$, arba $p_{k_p} = 1$, nagrinėjamo kintamojo reikšmė parenkama pagal (3.2) formulę.

5 punktas

Išdėstomos stotys pagal 3.3 skyrelio metodikas. Pagal priskirtas kintamųjų reikšmes apskaičiuojami kokybės rodikliai $Q_1, Q_2, .. Q_r$.

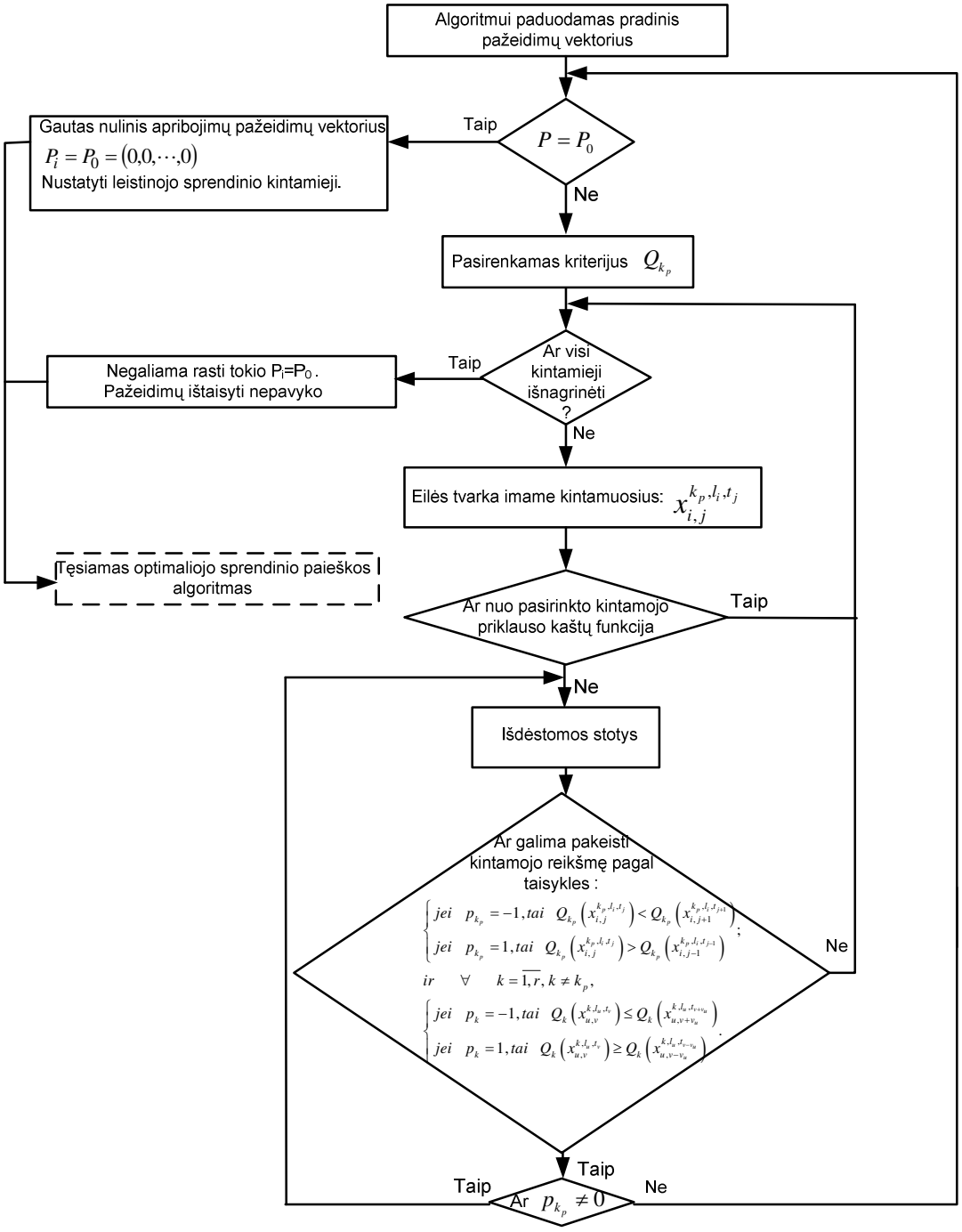
6 punktas

Randamas apribojimų pažeidimų vektorius $P = (p_1, p_2, \cdots, p_r)$, kuris apibrėžiamas (3.3) formule. Jei tenkinamos (2.17) sąlygos, grįžtama į 4 punktą.

7 punktas

Jei pažeista antra arba trečia iš (2.17) vykdymo sąlygų, kartojame 3 punktą. Jei pažeista pirmoji sąlyga, tuomet k_p-asis pažeidimas ištaisytas – kartojame 2 punktą.

Nubraižome pažeidimų taisymo algoritmo blokinę schemą:



3.3 pav. Pažeidimų taisymo algoritmo blokinė schema

3.3. STOČIŲ IŠDĖSTYMO ALGORITMAI

Šiame skyriuje pateiksime algoritmus, skirtus optimaliam stočių išdėstymui bei vartotojų prijungimui. Algoritmai skirsis priklausomai nuo pasirinkto prieigos tinklo tipo.

3.3.1. MOBILIOJO TELEFONO RYŠIO TINKLO MODELIAVIMAS

Duota:

- Teritorijos, kurioje reikia teikti mobiliojo ryšio paslaugą, žemėlapis. Ši teritorija yra suskirstyta pagal pasirinkto dydžio tinklelį - kvadratais. Tinklelio elementuose vartotojų skaičius pasiskirstęs tolygiai.
- Galimų stočių buvimo vietų žemėlapis. Teritorijos žemėlapio koordinačių rinkinys, kuris reprezentuoja galimas stočių statymo vietas.
- Stočių bei žemės kaštai atitinkamuose teritorijos žemėlapio taškuose.
- Teritorijos tinklelio elementai, kuriuos stotis gali aptarnauti būdama konkrečiame teritorijos taške.

Rasti:

Teritorijos žemėlapio koordinačių rinkinį, kuriuose pastatę stotis su pradiniuose duomenyse pateiktais parametrais padengsime norimą procentą teritorijos kiek įmanoma mažesniais kaštais.

Genetinis algoritmas

Sprendžiant stočių išdėstymo problemą, geną apibrėšime kaip vieną sprendinio vektoriaus (galimų stočių dislokacijos vietų) elementą:

$$genas_i = \begin{cases} 1, \ jei\ i-oji\ stotis\ \ pasirenkama \\ 0, \ jei\ i-oji\ stotis\ atmetama \end{cases} \qquad \textbf{(3.9)}$$

Chromosoma (sprendinys) - tai šių genų rinkinys, kuris vienareikšmiškai atstovauja kiekvieną populiacijos individą.

$$Chromosoma = \left(genas_1, ..., genas_n\right) \qquad \textbf{(3.10)}$$

n – galimų stočių dislokacijos vietų skaičius

Todėl turėdami chromosomą kokybės funkcijos pagalba galime įvertinti individą. Naudosime šią kokybės funkcijos išraišką:

$$f(Chromosoma) = \frac{\left(\frac{Aptarnaujama\ teritorija}{Visa\ teritorija}\right)^4}{\sum_{i \in N} C_i}, \qquad \textbf{(3.11)}$$

čia C_i - i-osios stoties kaštai, o $N = \{i | genas_i = 1\}$

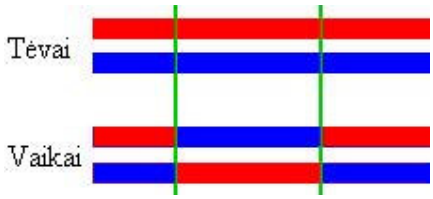
Realizuojant algoritmą, pradinės populiacijos chromosomas generuosime atsitiktinai. Taip pat naudosime įprastinius standartinei genetinio algoritmo schemai veiksmus: mutavimą, kryžminimą ir kitus. Po bet kurio veiksmo su populiacijos individais jie būtinai turi išlikti tinkamais sprendiniais

pagal šio uždavinio formuluotę. Priešingu atveju galime gauti sprendinį, kuris netenkins keliamų apribojimų.

Algoritmo evoliucionavimo schema

Reprodukcijos metu pirmiausiai išrenkama keletas pačių geriausių individų, kurie pateks į kitos epochos populiaciją. Tokios reprodukcijos privalumas prieš paprastą tikimybinę reprodukciją yra tas, jog geriausi populiacijos individai yra neprarandami perėjimo į kitą epochą metu. Bet, jei parametrai netinkamai parinkti, gali pasireikšti ir elitinės strategijos trūkumas - konvergavimas į lokalų ekstremumą. Todėl negalima piktnaudžiauti per didelėmis šio parametro reikšmėmis.

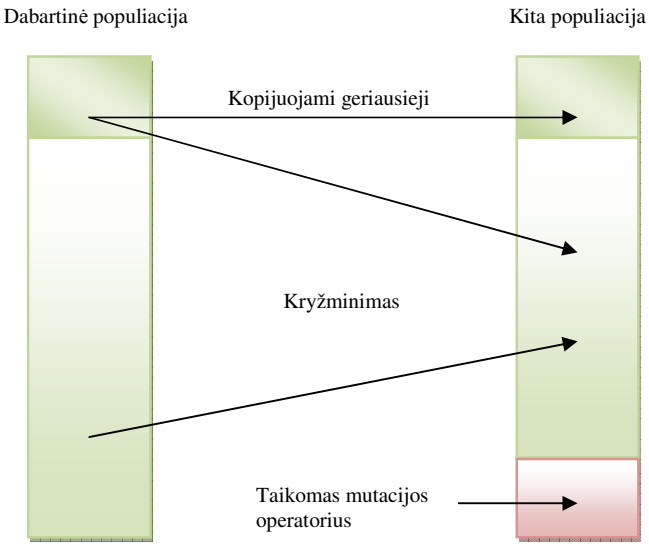
Algoritmo realizacijoje bus naudojamas dvitaškis kryžminimas. 3.4 paveiksle pateikiamas kryžminimo pavyzdys. Atsitiktinai parenkami kryžminimo taškai ir dvi dvejetainių vektorių (tėvų) dalys sukeičiamos vietomis gaunant du naujus dvejetainius vektorius (vaikus):



3.4 pav. Kryžminimo pavyzdys

Vietoj standartinės mutavimo procedūros, kai gali mutuoti kiekvieno individo kiekvienas genas pagal tam tikrą mažą tikimybę, naudosime mutavimo procedūrą, kurios metu iš populiacijos išrenkami blogiausi kokybės funkcijos prasme individai ir vietoje jų atsitiktinai sugeneruojami nauji.

Pateikiame vienos genetinio algoritmo iteracijos schemą:



3.5 pav. Principinė genetinio algoritmo vienos iteracijos schema

Pateikiame genetinio algoritmo pseudo kodą:

Genetinis Algoritmas {
  Generuoti pradinę populiaciją ***P***t
  Kol sustojimo kriterijus netenkinamas **Kartoti** {
    Išrinkti elementus iš ***P***t kurie bus kopijuojami į ***P***t+l
    Kryžminti elementus iš ***P***t ir įdėti į ***P***t+l
    Mutuoti elementus iš ***P***t ir įdėti į ***P***t+l
    Taikyti lokalios paieškos procedūrą
    Pt = ***P***t+l
  }
}

Lokalios paieškos procedūra

Norint pagreitinti sprendinio gerėjimą, tikslinga genetinį algoritmą hibridizuoti įtraukiant lokalios paieškos procedūrą. Lokalios paieškos procedūra – tai algoritmas, kuris pakeičia populiacijos individus reprezentuojančias chromosomas taip, kad f(Chromosoma) ≤ f(Chromosoma*),

čia: Chromosoma* = „**Lokalios Paieškos Procedūra“**(Chromosoma) **(3.12)**

Pateikiame Lokalios paieškos procedūros pseudo kodą:

Lokalios Paieškos procedūra {
  Nuo i = 1 **Iki** stočių skaičius **Kartoti** {
    Ar $genas_i = 1$ {
      $Chromosoma^* = (genas_1, ..., genas_{i-1}, 0, genas_{i+1}, ..., genas_n)$
      Ar („teritorijos padengimo procentas“(Chromosoma) =
        „teritorijos padengimo procentas“(Chromosoma*))
        $genas_i = 0$
      Kitais atvejais
        $Chromosoma^* = Chromosoma$
    }
  }
}

Skruzdžių kolonijos algoritmas

Duota:

$A = (a_{ij})$ A – $m \times n$ matavimų matrica, kur $a_{ij} = \begin{cases} 1, & jei\ j\ stulpelis\ padengia\ i\ eilutę \\ 0, & kitais\ atvejais \end{cases}$

$\forall j := b_j, b_j \geq 0$ b_j – kiekvienam stulpeliui priskirti stoties pastatymo kaštai

Rasti:

$$f(y) = \sum_{j=1}^{n} b_j \cdot y_j \rightarrow \min \quad \textbf{(3.13)}$$

kai galioja apribojimai: $\sum_{j=1}^{n} a_{ij} \cdot y_j \geq k, \ i \in N^m$ **(3.14)**

N^m - aibė teritorijos tinklelio elementų, kuriuose gyventojų skaičius didesnis už 0;

k - nustatomas pagal paslaugos vartotojų skaičių teritorijos tinklelio elemente;

$y_j \in \{0, 1\}, \ j = \overline{1, n}$ $y_j = \begin{cases} 1, \textit{ jei } j-\textit{oji kolona priklauso sprendiniui} \\ 0, \textit{ kitais atvejais} \end{cases}$

Čia matrica A – „teritorijos žemėlapis“, kur eilutės atitinka teritorijos tinklelio elementus, o stulpeliai atitinka galimas siųstuvų dislokacijos vietas. a_{ij} parodo ar i-ąjį teritorijos tinklelio elementą gali aptarnauti j-asis siųstuvas. Tad, matrica A vienareikšmiškai nurodo teritorijos žemėlapį ir siųstuvų aptarnaujamas teritorijas. Apribojimų sistema garantuoja, kad absoliučiai visa teritorija yra padengiama. Jei turime teritoriją, kurioje ryšio poreikis yra netolygus, tai keisdami apribojimų sistemą nesunkiai algoritmą modifikuosime. Matome, kad tokia uždavinio matematinė formuluotė yra ekvivalenti problemos formuluotei.

Pateikiame „Skruzdžių kolonijos“ algoritmo pseudo kodą:

„Skruzdžių kolonijos” algoritmas {
Nustatyti pradinius parametrus
Kol neįvykdytas numatytas iteracijų skaičius **Kartoti** {
Nuo k = 1 **Iki** „skruzdžių” skaičius **Kartoti** {
Kol sprendinys nesukonstruotas **Kartoti** {
Įtraukti stulpelį į sprendinį
Įtraukti stulpelį į „tabu” sąrašą
}
Taikyti lokalios paieškos procedūrą
}
Atnaujinti geriausią sprendinį
Atnaujinti „feromonus” kiekvienam stulpeliui
}
}

čia:

- Pradinių parametrų parinkimas apima „skruzdžių“ skaičiaus nustatymą, o svarbiausia - taisyklės, pagal kurią sprendžiama, kurį stulpelį traukti į sprendinį, o kurio ne;
- Algoritmo stabdymo strategijų gali būti įvairių: fiksuojamas iteracijų skaičius, kartojama tol, kol geriausias sprendinys nebegerėja ir pan.;
- Ar stulpelis bus traukiamas į sprendinį, yra nustatoma atsitiktinių skaičių generatoriaus pagalba. Tikimybė stulpeliui patekti į sprendinį labiausiai priklauso nuo dviejų veiksnių: p = p(euristinė informacija, „feromonų” informacija);
- „Feromonų“ atnaujinimas - tai procesas, kurio metu stulpeliams priskiriama informacija apie jų „populiarumą“ renkant juos į sprendinius.

Tikimybė patekti į sprendinį

$$p_i^k(t) = \frac{[\tau_i(t)] \cdot [\eta_i]^\beta}{\sum_{l \in N^k} [\tau_l(t)] \cdot [\eta_l]^\beta}, \text{ jei } i \in N^k \qquad \textbf{(3.15)}$$

čia:

- k – „skruzdės“ numeris.
- i – nagrinėjamo stulpelio numeris.
- t – iteracijos numeris.
- τ_j – „feromonų“ informacija j-ajam stulpeliui.
- η_j – euristinė informacija j-ajam stulpeliui. $\eta_j = \frac{d_j}{b_j}$, **(3.16)**

  kur $d_j = \sum_{i=1}^{n} a_{ij}$ - j-osios stoties padengiamų teritorijų tinklelio elementų skaičius

  b_j - j-osios stoties pastatymo kaštai.
- N^k – leistinų k-ajai „skruzdei“ stulpelių, kurie padengia bent po vieną dar nepadengtą matricos A eilutę, aibė.

j-ojo stulpelio „feromonų“ atnaujinimo taisyklė

$$\tau_j = (1 - \rho) \cdot \tau_j + \rho \cdot \Delta\tau_j \qquad \textbf{(3.17)}$$

čia:

- ρ – „išgaravimo“ koeficientas (nurodomas kaip algoritmo parametras).
- Δτ – j-ojo stulpelio dažnis „skruzdžių“ suformuotuose sprendiniuose

Lokalios paieškos procedūra

Siekiant pagreitinti sprendinio gerėjimą, tikslinga į „Skruzdžių kolonijos“ algoritmą įtraukti taip vadinamus „demoniškus veiksmus“, arba kitaip - lokalios paieškos procedūrą. Lokalios paieškos procedūra – tai algoritmas, kuris nemažindamas aptarnaujamos teritorijos procento pašalina perteklines stotis. Akivaizdu, kad

$$\sum_{i \in N^m} C_i \le \sum_{i \in N^n} C_i \qquad \textbf{(3.18)}$$

čia C_i - i-osios stoties kaštai;

N^n - stočių, priklausančių sprendiniui, aibė;

N^m - stočių, priklausančių sprendiniui, aibė, gauta sprendinį paveikus lokalios paieškos procedūra;

Pateikiame Lokalios paieškos procedūros pseudo kodą:

Lokalios Paieškos procedūra {

$N^m = N^n$

Nuo i = 1 **Iki** card{ N^n } **Kartoti**

Ar („teritorijos padengimo procentas“($N^m \setminus i$) =

„teritorijos padengimo procentas“(N^m))

$N^m = N^m \setminus i$

}

3.3.2. FIKSUOTO TELEFONO RYŠIO TINKLO MODELIAVIMAS

Duota:

- Teritorijos, kurioje reikia teikti fiksuoto telefono ryšio paslaugą, žemėlapis. Ši teritorija yra suskirstyta pagal pasirinkto dydžio tinklelį - kvadratais. Tinklelio elementuose vartotojų skaičius pasiskirstęs tolygiai.
- Galimų stočių buvimo vietų žemėlapis. Teritorijos žemėlapio koordinačių rinkinys, kuris reprezentuoja galimas stočių statymo vietas.
- Stočių bei žemės kaštai atitinkamuose teritorijos žemėlapio taškuose.
- Vartotojo prijungimo kaštai kilometrui.
- Teritorijos tinklelio elementai, kuriuos stotis gali aptarnauti būdama konkrečiame teritorijos taške.

Rasti:

Teritorijos žemėlapio koordinačių rinkinį, kuriuose pastatę stotis su pradiniuose duomenyse pateiktais parametrais prie stočių galėsime prijungti norimą procentą teritorijos kiek įmanoma mažesniais kaštais.

Genetinis algoritmas

Sprendžiant stočių išdėstymo problemą, geną apibrėšime kaip vieną teritorijos tinklelio vektoriaus elementą:

$$genas_i = \begin{cases} 1, \ jei\ i-asis\ teritorijos\ tinklelio\ elementas\ prijungiamas \\ 0, \ jei\ i-asis\ teritorijos\ tinklelio\ elementas\ neprijungiamas \end{cases} \quad \textbf{(3.19)}$$

Chromosoma - tai šių genų rinkinys, kuris vienareikšmiškai atstovauja konkrečiai stočiai priskirtus teritorijos tinklelio elementus.

$$Chromosoma_j = (genas_1, ..., genas_n) \quad \textbf{(3.20)}$$

n – teritorijos tinklelio elementų skaičius

j – j-oji potenciali dislokacijos vieta

Individas (sprendinys) - tai šių chromosomų rinkinys:

$$Individas = (Chromosoma_1, ..., Chromosoma_m) \quad \textbf{(3.21)}$$

m – potencialių stočių dislokacijos vietų skaičius

Turėdami individą, kokybės funkcijos pagalba galime įvertinti populiacijos individą. Naudosime tokią kokybės funkcijos išraišką:

$$f(Individas) = \frac{\left(\frac{Aptarnaujama\ teritorija}{Visa\ teritorija}\right)^4}{\sum_{i \in N}(C_i + D_i)}, \quad \textbf{(3.22)}$$

čia C_i - i-osios stoties pastatymo kaštai;

D_i - abonentų prijungimo kaštai i-ajai stočiai;

$$N = \left\{ i \left| \sum_{j=1}^{n} genas_j \geq 1 \right. \right\}$$

Realizuojant algoritmą, taip bus naudojamas atsitiktinis individo generatorius. Taip pat bus atliekami mutavimo, kryžminimo ir kiti veiksmai. Po bet kurio veiksmo su populiacijos individais jie būtinai turi išlikti tinkamais sprendiniais pagal konkretaus uždavinio formuluotę, priešingu atveju galime gauti sprendinį, kuris netenkins keliamų apribojimų.

Algoritmo evoliucionavimo schema tokia pat kaip ir genetinio algoritmo mobiliojo telefono ryšio tinklui pateikta 33 puslapyje.

Lokalios paieškos procedūra

Norint pagreitinti sprendinio gerėjimą, tikslinga Genetinį algoritmą hibridizuoti įtraukiant lokalios paieškos procedūrą. Lokalios paieškos procedūrą – tai algoritmas, kuris pakeičia populiacijos individus reprezentuojančias chromosomas taip, kad f(Individas) ≤ f(Individas*),

čia: Individas* = „**Lokalios Paieškos Procedūra“**(Individas) **(3.23)**

Pateikiame Lokalios paieškos procedūros pseudo kodą:

Lokalios Paieškos procedūra {
  Nuo i = 1 **Iki** stočių skaičius **Kartoti**
    Taikyti teritorijos tinklelio elementų perskirstymo procedūrą
  Nuo i = 1 **Iki** stočių skaičius **Kartoti {**
    Ar $\sum_{j=1}^{n} genas_j \geq 1$ {
      $Chromosoma^* = (0,...,0)$
      $Individas^* = \left(Chromosoma_1,...,Chromosoma^*,...,Chromosoma_m\right)$
      Ar („teritorijos padengimo procentas“(Individas) =
        „teritorijos padengimo procentas“(Individas*))
        $Chromosoma_i = Chromosoma^*$
      Kitais atvejais
        $Chromosoma^* = Chromosoma$
    }
  }
}

Skruzdžių kolonijos algoritmas

Duota:

$A = (a_{ij})$ A – $m \times n$ matavimų matrica, kur $a_{ij} = \begin{cases} 1, & jei\ j\ stulpelis\ padengia\ i\ eilutę \\ 0, & kitais\ atvejais \end{cases}$

$\forall j := b_j, b_j \geq 0$ b_j – kiekvienam stulpeliui, priskirti stoties pastatymo kaštai

Rasti:

$$f(y) = \sum_{j=1}^{n} b_j \cdot y_j \rightarrow \min \qquad \textbf{(3.24)}$$

kai galioja apribojimai: $\sum_{j=1}^{n} a_{ij} \cdot y_j \geq 1, \ \ i \in N^m$ **(3.25)**

N^m - aibė teritorijos tinklelio elementų, kuriuose gyventojų skaičius didesnis už 0;

$y_j \in \{0, 1\},\ j = \overline{1, n}$ $\quad y_j = \begin{cases} 1, \textit{ jei } j-\textit{oji kolona priklauso sprendiniui} \\ 0, \textit{ kitais atvejais} \end{cases}$

Čia matrica A – „teritorijos žemėlapis“, kur eilutės atitinka teritorijos tinklelio elementus, o stulpeliai atitinka galimas siųstuvų dislokacijos vietas. a_{ij} parodo, ar i-ąjį teritorijos tinklelio elementą gali aptarnauti j-asis siųstuvas. Matrica A vienareikšmiškai nusako teritorijos žemėlapį ir siųstuvų aptarnaujamas teritorijas. Apribojimų sistema garantuoja, kad absoliučiai visa teritorija, kurioje reikalingas ryšys, yra padengiama. Matome, kad tokia uždavinio matematinė formuluotė yra ekvivalenti problemos formuluotei. Algoritmo pseudo-kodas toks pat kaip ir „skruzdžių kolonijos“ algoritmo mobiliojo telefono ryšio tinklui pateiktas 35 puslapyje.

Tikimybė patekti į sprendinį

$$p_i^k(t) = \frac{[\tau_i(t)] \cdot [\eta_i]^\beta}{\sum_{l \in N^k} [\tau_l(t)] \cdot [\eta_l]^\beta}, \text{ jei } i \in N^k \qquad \textbf{(3.26)}$$

čia:

- k – „skruzdės“ numeris.
- i – nagrinėjamo stulpelio numeris.
- t – iteracijos numeris.
- τ_j – „feromonų“ informacija j-ajam stulpeliui.
- η_j – euristinė informacija j-ajam stulpeliui. $\eta_j = \frac{e_j}{b_j + c_j}$, **(3.27)**

  kur $e_j = \sum_{i \in N^n} a_{ij}$ - j-tosios stoties padengiamų teritorijų tinklelio elementų skaičius

  N^n – teritorijos tinklelio elementai, tenkinantys bent vieną iš šių sąlygų:

  - tinklelio elementas neprijungtas prie jokios sprendinyje esančios stoties;
  - tinklelio elemento prijungimo kaštai prie j-osios stoties t_{ij} yra mažesni už prijungimo kaštus prie bet kurios kitos sprendiniui priklausančios stoties:

    $t_{ij} = \min_j t_i(j)$ **(3.28)**

  b_j - j-osios stoties pastatymo kaštai;

  c_j - abonentų prijungimo kaštai j-ajai stočiai;

- N^k – leistinų k-ajai „skruzdei“ stulpelių aibė, kurie padengia bent po vieną dar nepadengtą matricos A eilutę.

j-ojo stulpelio „feromonų” atnaujinimo taisyklė apibrėžiama 3.17 formule.

Lokalios paieškos procedūra

Norint pagreitinti sprendinio gerėjimą, tikslinga į „Skruzdžių kolonijos“ algoritmą įtraukti taip vadinamus „demoniškus veiksmus“ arba kitaip - lokalios paieškos procedūrą. Lokalios paieškos procedūra – tai algoritmas, kuris nemažindamas aptarnaujamos teritorijos procento pašalina perteklines stotis. Akivaizdu, kad

$$\sum_{i \in N^m} C_i \le \sum_{i \in N^n} C_i \qquad \textbf{(3.29)}$$

čia C_i - i-osios stoties kaštai;

N^n - stočių, priklausančių sprendiniui, aibė;

N^m - stočių, priklausančių sprendiniui, aibė, gauta sprendinį paveikus lokalios paieškos procedūra;

Pateikiame Lokalios paieškos procedūros pseudo kodą:

Lokalios Paieškos procedūra {

$N^m = N^n$

Nuo i = 1 **Iki** card{ N^n } **Kartoti**

Ar ((**„**teritorijos padengimo procentas“($N^m \setminus i$) =

„teritorijos padengimo procentas**“**(N^m)) **ir**

($\sum_{j \in N^m \setminus i} C_j \le \sum_{i \in N^m} C_j$))

$N^m = N^m \setminus i$

}

3.4. TYRIMAI IR REZULTATAI

Visų tyrimų metu skaitmeniniai teritorijos ir potencialių stočių buvimo vietų žemėlapiai yra pastovūs, sugeneruoti sukurtos programinės įrangos pagal šiuos parametrus:

- Teritorijos elementų skaičius – 400;
- Potencialių stočių dislokacijos vietų skaičius – 49;
- Žemės kaina taškuose sugeneruojama atsitiktinai pagal Tolygųjį skirstinį intervale (0; 100000);
- Stoties pastatymo kaina – 30000 Lt;
- Stoties aprėpties spindulys – 30 km;
- Vartotojo prijungimo kaštai 1 km – 0.01 Lt.

Bylos su minėtais duomenimis pateikiamos kompaktiniame diske, kuris yra prisegtas prie šio darbo.

3.4.1. STOČIŲ IŠDĖSTYMO ALGORITMŲ REZULTATŲ TYRIMAI

Šiame skyrelyje pateiksime stočių išdėstymo algoritmų skaičiavimų rezultatus. Taipogi panagrinėsime šių rezultatų priklausomybę nuo pagrindinių algoritmų parametrų. Tai atlikti yra svarbu, nes nuo teisingo parametrų parinkimo didžiąja dalimi priklauso algoritmų rezultatų kokybė ir jų tinkamumas užduočiai spręsti. Tyrimus apibendrinančius samprotavimus pateiksime 5 skyriuje.

3.4.1.1. SKRUZDŽIŲ KOLONIJOS ALGORITMO PARAMETRŲ TYRIMAI

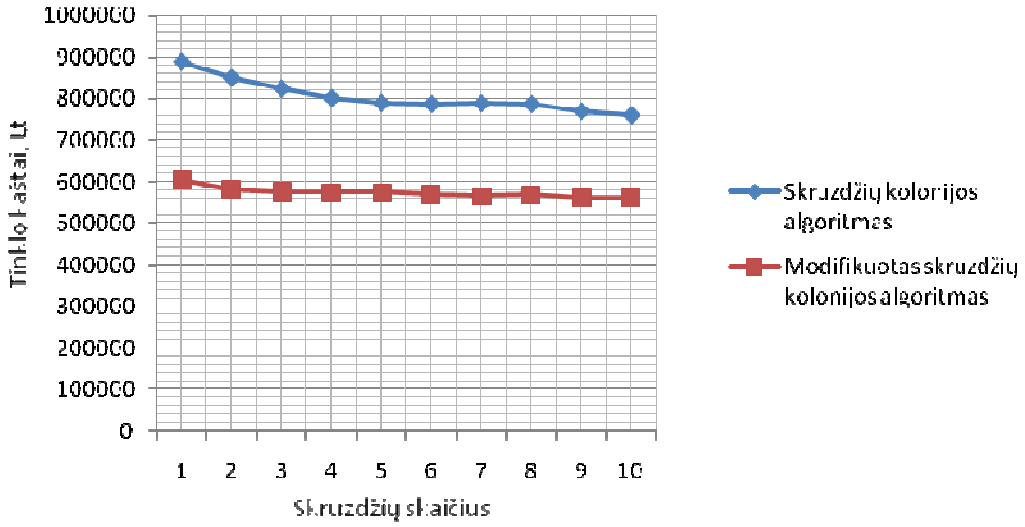
„Skruzdžių“ skaičiaus parametro tyrimas

„Skruzdžių“ skaičius – svarbus „Skruzdžių kolonijos“ algoritmo parametras. Nuo šio parametro reikšmės priklauso tikimybės rasti geriausią sprendinį dydis bei algoritmo vykdymo „kaštai“ kompiuterio resursų prasme.

Pateikiame apskaičiuotų kaštų bei minėto „skruzdžių“ skaičiaus parametro priklausomybės grafiką, kai likusiem Skruzdžių kolonijos algoritmo parametrams priskirtos tokios reikšmės:

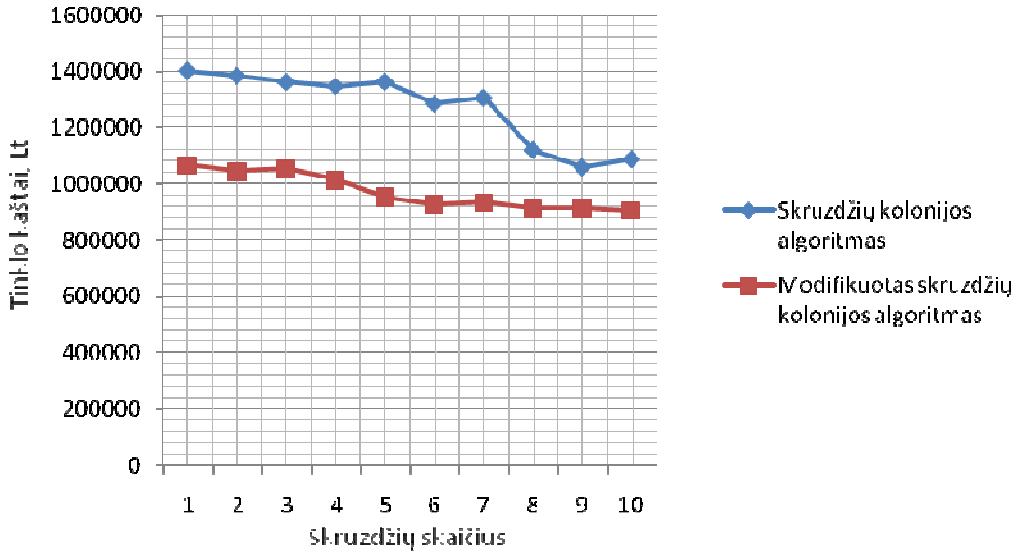
- Iteracijų skaičius = 5
- $\beta = 0{,}4$
- $\rho = 0{,}5$

Mobiliojo telefono ryšio tinklui



3.6 pav. „Skruzdžių kolonijos“ algoritmo „skruzdžių“ skaičiaus tyrimas

Fiksuoto telefono ryšio tinklui



3.7 pav. „Skruzdžių kolonijos“ algoritmo „skruzdžių“ skaičiaus tyrimas 2

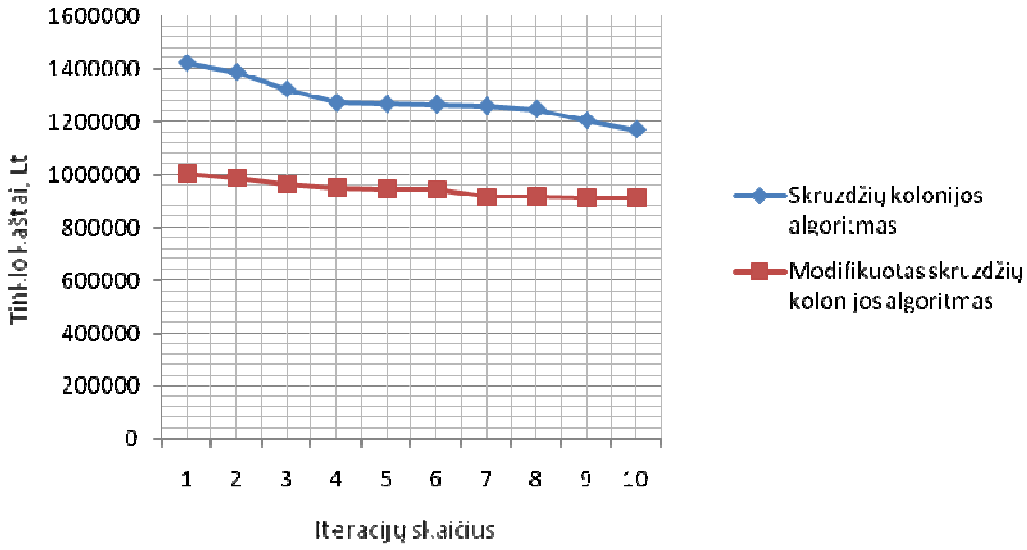
Iteracijų skaičiaus parametro tyrimas

Iteracijų skaičius – nemažiau svarbus Skruzdžių kolonijos algoritmo parametras. Nuo šio parametro reikšmės priklauso tikimybės rasti geriausią sprendinį dydis bei algoritmo vykdymo „kaštai“ kompiuterio resursų prasme. Didesnės šio parametro reikšmės leidžia algoritmo „skruzdėms“ tinkamiau išnaudoti „feromonų“ informacijos teikiamus privalumus.

Pateikiame apskaičiuotų kaštų bei iteracijų skaičiaus parametro priklausomybės grafiką, kai likusiems „Skruzdžių kolonijos“ algoritmo parametrams priskirtos tokios reikšmės:

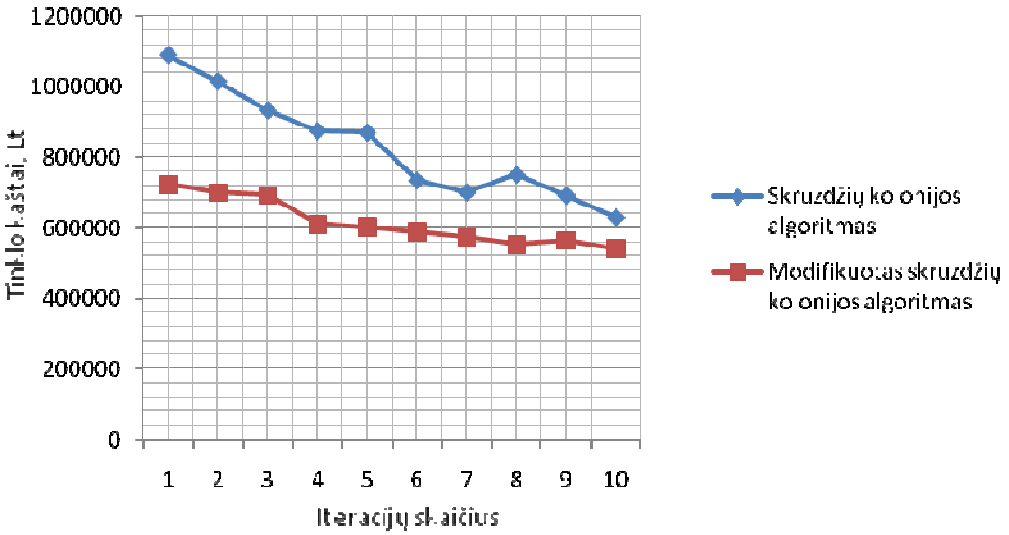
- „Skruzdžių“ skaičius = 5
- $\beta = 0,4$
- $\rho = 0,5$

Fiksuoto telefono ryšio tinklui



3.8 pav. „Skruzdžių kolonijos“ algoritmo iteracijų skaičiaus tyrimas

Mobiliojo telefono ryšio tinklui



3.9 pav. „Skruzdžių kolonijos“ algoritmo iteracijų skaičiaus tyrimas 2

3.4.1.2. GENETINIO ALGORITMO PARAMETRŲ TYRIMAI

Visais toliau pateikiamų skaičiavimų atvejais padengtos teritorijos procentas ≥ 95 %

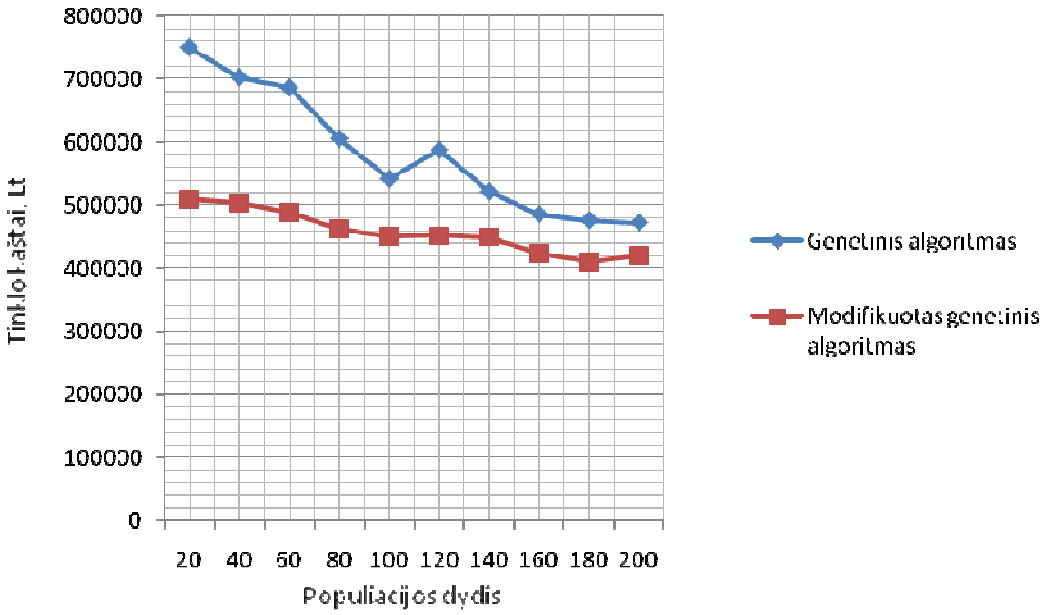
Populiacijos dydžio parametro tyrimas

Populiacijos dydis – svarbus Genetinio algoritmo parametras. Nuo šio parametro reikšmės priklauso tikimybės rasti geriausią sprendinį dydis bei algoritmo vykdymo „kaštai“ kompiuterio resursų prasme. Didesnės šio parametro reikšmės padidina sprendinių kandidatų skaičių, dėl to padidėja algoritmo pasirinkimo galimybė.

Pateikiame apskaičiuotų kaštų bei minėto populiacijos dydžio parametro priklausomybės grafiką, kai likusiems Genetinio algoritmo parametrams priskirtos tokios reikšmės:

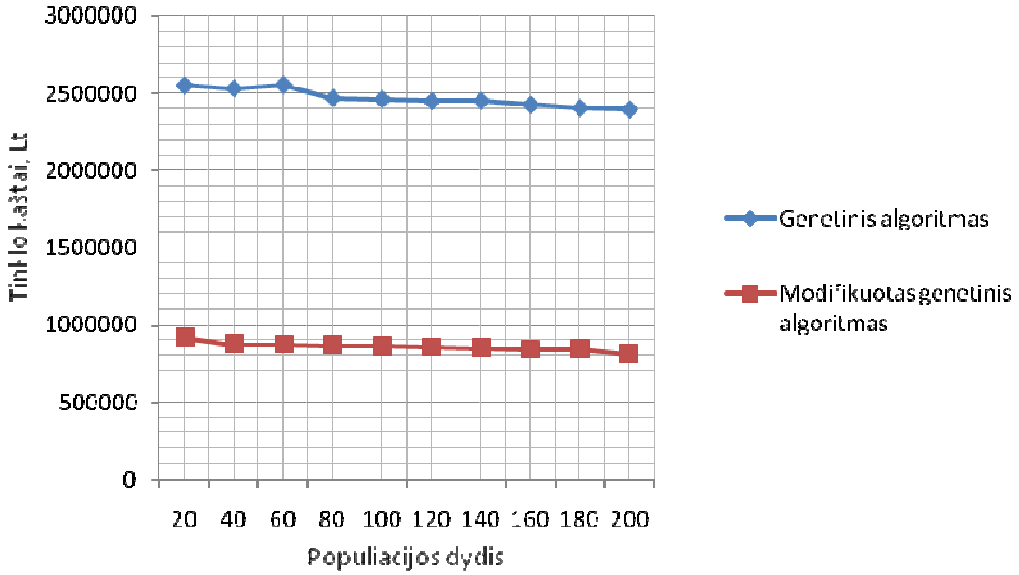
- Iteracijų skaičius = 20
- Mutavimo koef. = 0,1
- Elito koef. = 0,1

Mobiliojo telefono ryšio tinklui



3.10 pav. Genetino algoritmo populiacijos dydžio tyrimas

Fiksuoto telefono ryšio tinklui



3.11 pav. Genetino algoritmo populiacijos dydžio tyrimas 2

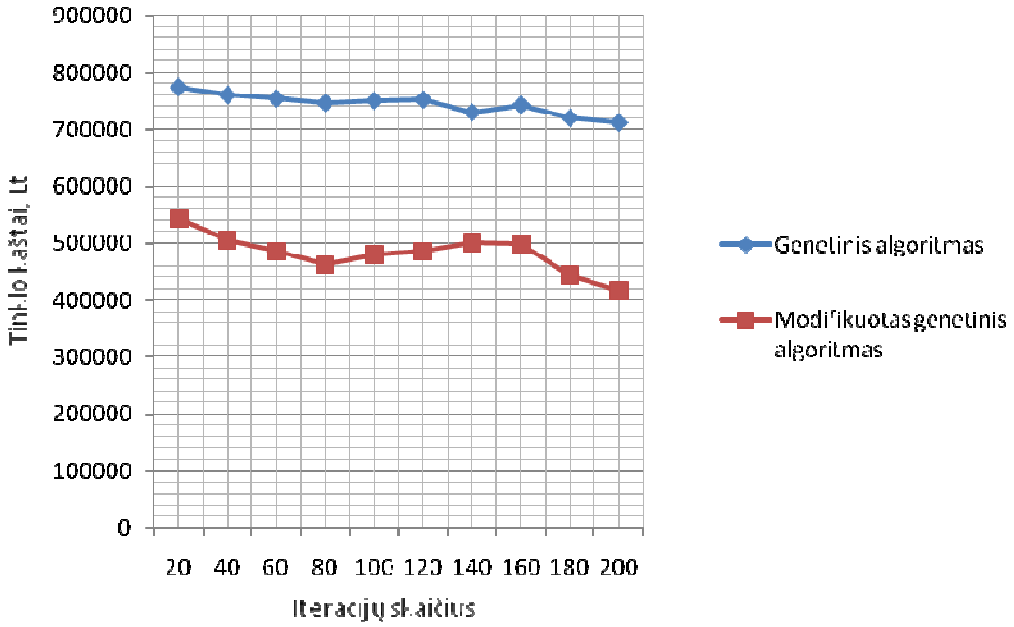
Iteracijų skaičiaus parametro tyrimas

Iteracijų skaičius – svarbus Genetinio algoritmo parametras. Nuo šio parametro reikšmės priklauso tikimybės rasti geriausią sprendinį dydis bei algoritmo vykdymo „kaštai“ kompiuterio resursų prasme. Didesnės šio parametro reikšmės leidžia algoritmo evoliuciniams procesams geriau atskleisti savo veiklos rezultatus ir taip pastebėti jų trūkumus/ privalumus.

Pateikiame apskaičiuotų kaštų bei minėto populiacijos dydžio parametro priklausomybės grafiką, kai likusiems Genetinio algoritmo parametrams priskirtos tokios reikšmės:

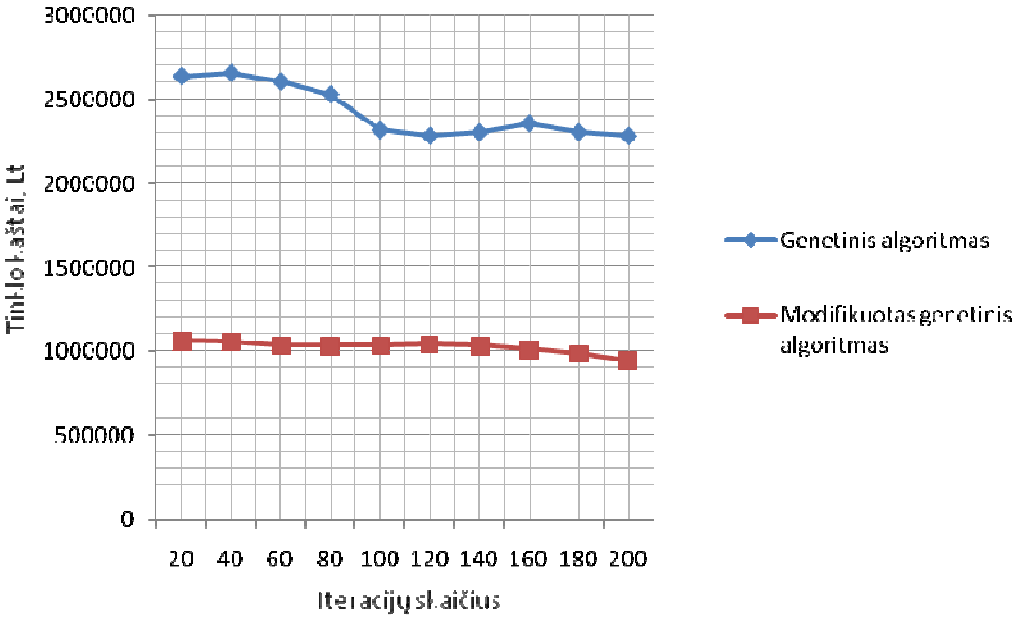
- Populiacijos dydis = 20
- Mutavimo koef. = 0,1
- Elito koef. = 0,1

Mobiliojo telefono ryšio tinklui



3.12 pav. Genetino algoritmo iteracijų skaičiaus tyrimas

Fiksuoto telefono ryšio tinklui



3.13 pav. Genetino algoritmo iteracijų skaičiaus tyrimas 2

3.4.2. DAUGIAPARAMETRINIO PRIEIGOS TINKLO OPTIMIZAVIMO ALGORITMO REZULTATAI

Dėl didelės modelio duomenų apimties, duomenys yra pateikti prieduose.

Programos nustatymai:

- Kintamieji ir jų galimos reikšmės pateiktos 1 priede;
- Kokybės apribojimai bei jų reikšmės:

$$\begin{cases} Q_{1\min} = 0 \\ Q_{1\max} = 30 \end{cases} \text{ ir } \begin{cases} Q_{2\min} = 3.54 \\ Q_{2\max} = 100 \end{cases}$$

- Apribojimų priklausomybės nuo kintamųjų:
  - $Q_1 = f(R, G)$, reikšmės pateikiamos 1 priede;
  - $Q_2 = f(L, R)$, reikšmės pateikiamos 1 priede;
  - Naudojama tokia kaštų funkcijos išraiška: $C = f(L, R)$, reikšmės pateikiamos 1 priede;
- Pradinis reikšmių rinkinys fiksuojamas atsitiktinai;
- Kintamųjų, kokybės kriterijų ir kaštų funkcijos reikšmės po kiekvienos iteracijos pateikiamos 2 priede;

Optimizavimo rezultatai fiksuoto telefono ryšio tinklui, kai stotims išdėstyti naudojamas modifikuotas Genetinis algoritmas, pateiktas 3.3.2 skyrelyje.

Stočių išdėstymo algoritmo parametrų reikšmės:

- Populiacijos dydis = 200
- Iteracijų skaičius = 20
- Mutavimo koef. = 0,1
- Elito koef. = 0,1
- Minimalus leistinas teritorijos padengimas – 90 %
- Algoritmo kartojimų skaičius – 1 kartas

Gautas sprendinys X^* = **{40; Radijo; MM1S, TDMA, QPSK, 832, 1};**

Šiame taške:

$Q_1(X^*)$ = **29.513 s.** ir $Q_2(X^*)$ = **29.132,** taigi tenkinama apribojimų sistema:

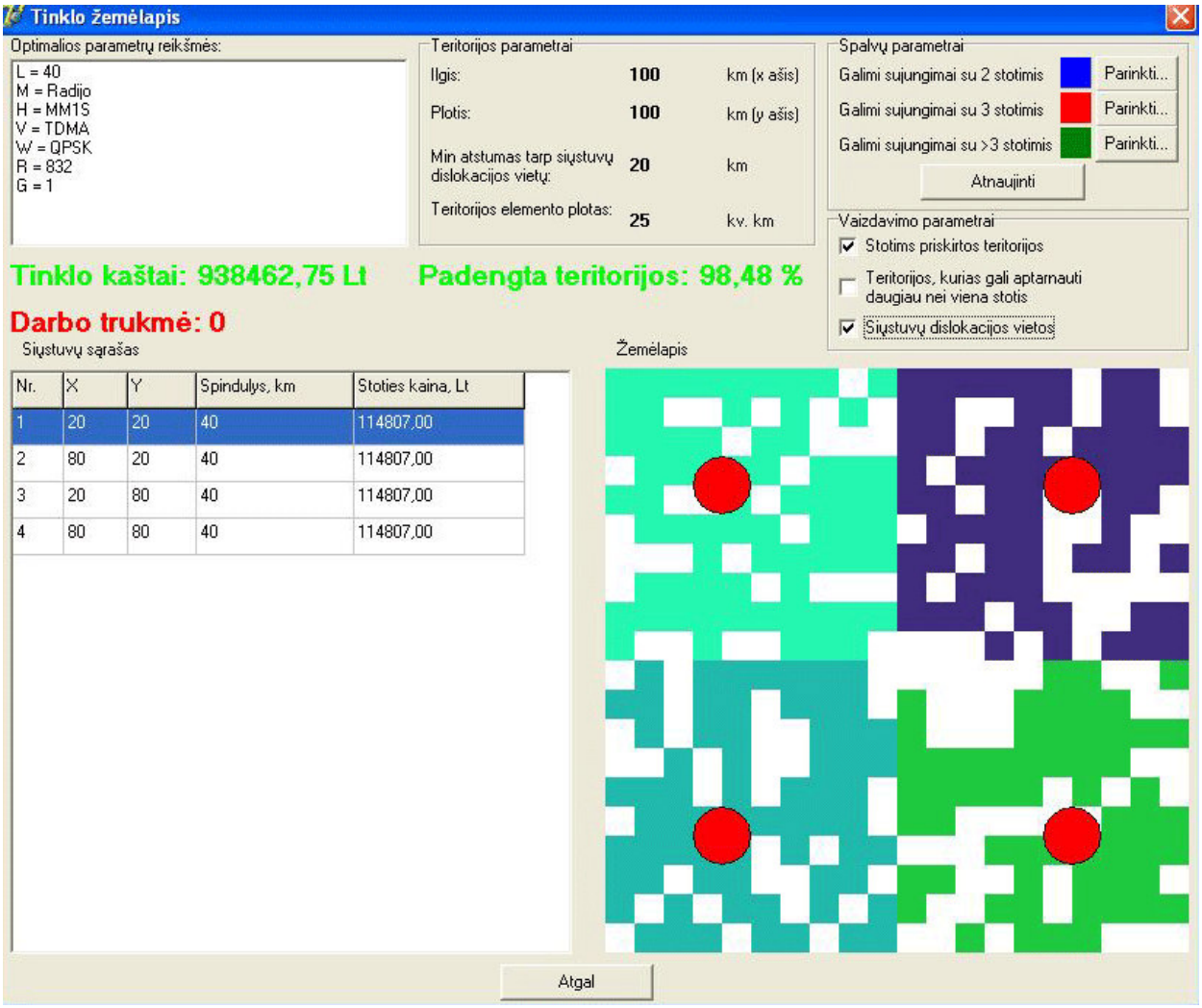
$$\begin{cases} 0 \le Q_1(X^*) \le 30 \\ 3.54 \le Q_2(X^*) \le 100 \end{cases}$$

X^* parametrais nusakomo tinklo diegimo kaštai yra:

$C_{min} = f(X^*)$ = **938462,75 Lt.**

Teritorijos padengimo procentas – **98,48 %**

Programos rezultatų lango vaizdas:



3.16 pav. Optimizavimo rezultatai fiksuoto telefono ryšio tinklui

Optimizavimo rezultatai fiksuoto telefono ryšio tinklui, kai stotims išdėstyti naudojamas modifikuotas „Skruzdžių kolonijos“ algoritmas, pateiktas 3.3.2 skyrelyje.

Stočių išdėstymo algoritmo parametrų reikšmės:

- Iteracijų skaičius = 5
- „Skruzdžių“ skaičius = 5
- $\beta = 0{,}4$
- $\rho = 0{,}5$
- Algoritmo kartojimų skaičius – 1 kartas

Gautas sprendinys $X^* =$ **{45; Radijo; MM1S, TDMA, 4QAM, 832, 1};**

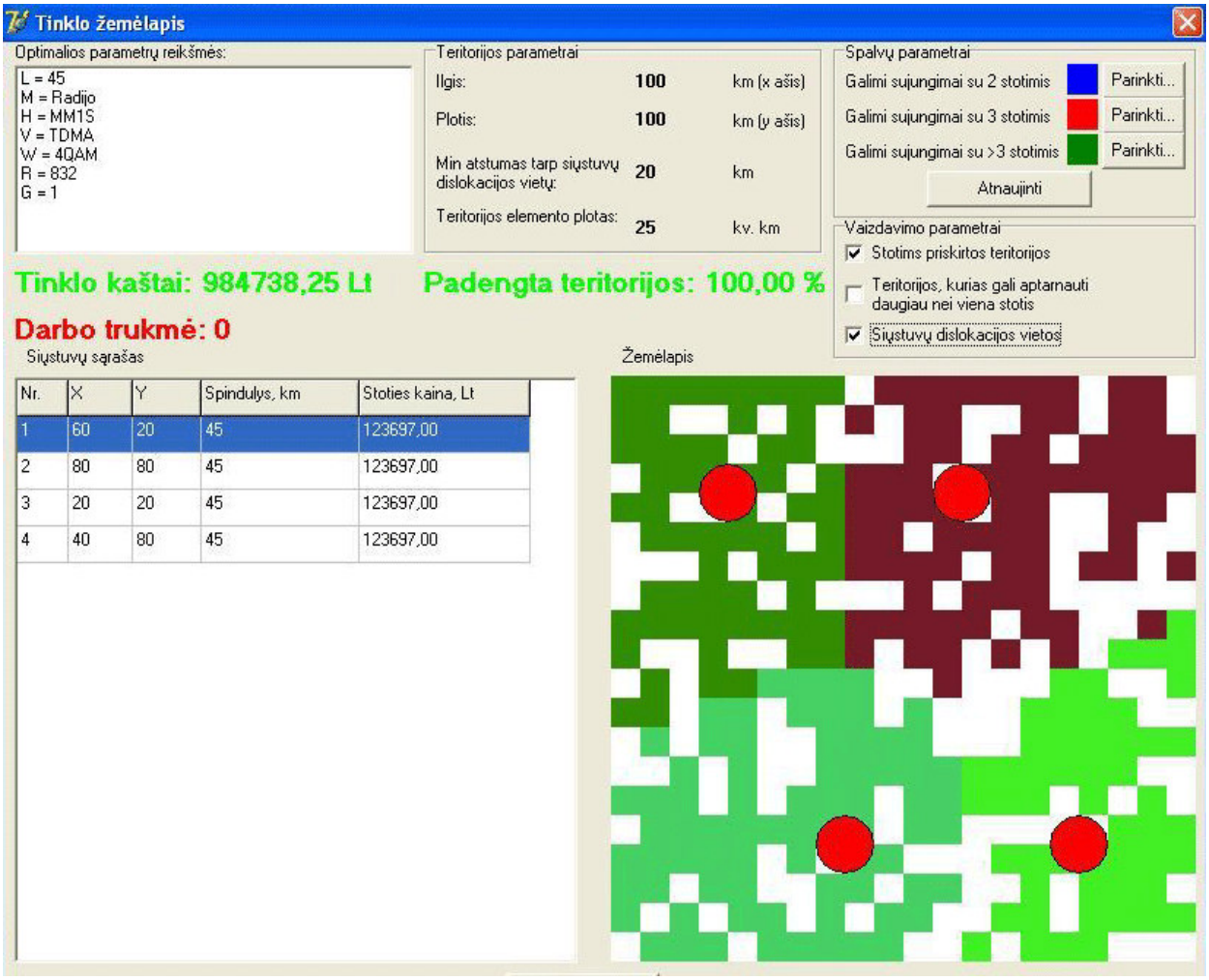
Šiame taške: $Q_1(X^*) =$ **29.513 s.** ir $Q_2(X^*) =$ **33.411,** taigi tenkinama apribojimų sistema:

$$\begin{cases} 0 \le Q_1(X^*) \le 30 \\ 3.54 \le Q_2(X^*) \le 100 \end{cases}$$

X^* parametrais nusakomo tinklo diegimo kaštai yra: C_{min}= f (X^*) = **984738,01 Lt.**

Teritorijos padengimo procentas – **100,00 %**

Programos rezultatų lango vaizdas:



3.17 pav. Optimizavimo rezultatai fiksuoto telefono ryšio tinklui 2

Optimizavimo rezultatai mobiliojo telefono ryšio tinklui, kai stotims išdėstyti naudojamas modifikuotas Genetinis algoritmas, pateiktas 3.3.1 skyrelyje.

Stočių išdėstymo algoritmo parametrų reikšmės:

- Populiacijos dydis = 20
- Iteracijų skaičius = 20
- Mutavimo koef. = 0,1
- Elito koef. = 0,1
- Minimalus leistinas teritorijos padengimas – 90 %
- Algoritmo kartojimų skaičius – 3 kartai

Gautas sprendinys X^* = **{45; Radijo; MM1S, TDMA, QPSK, 896, 2};**

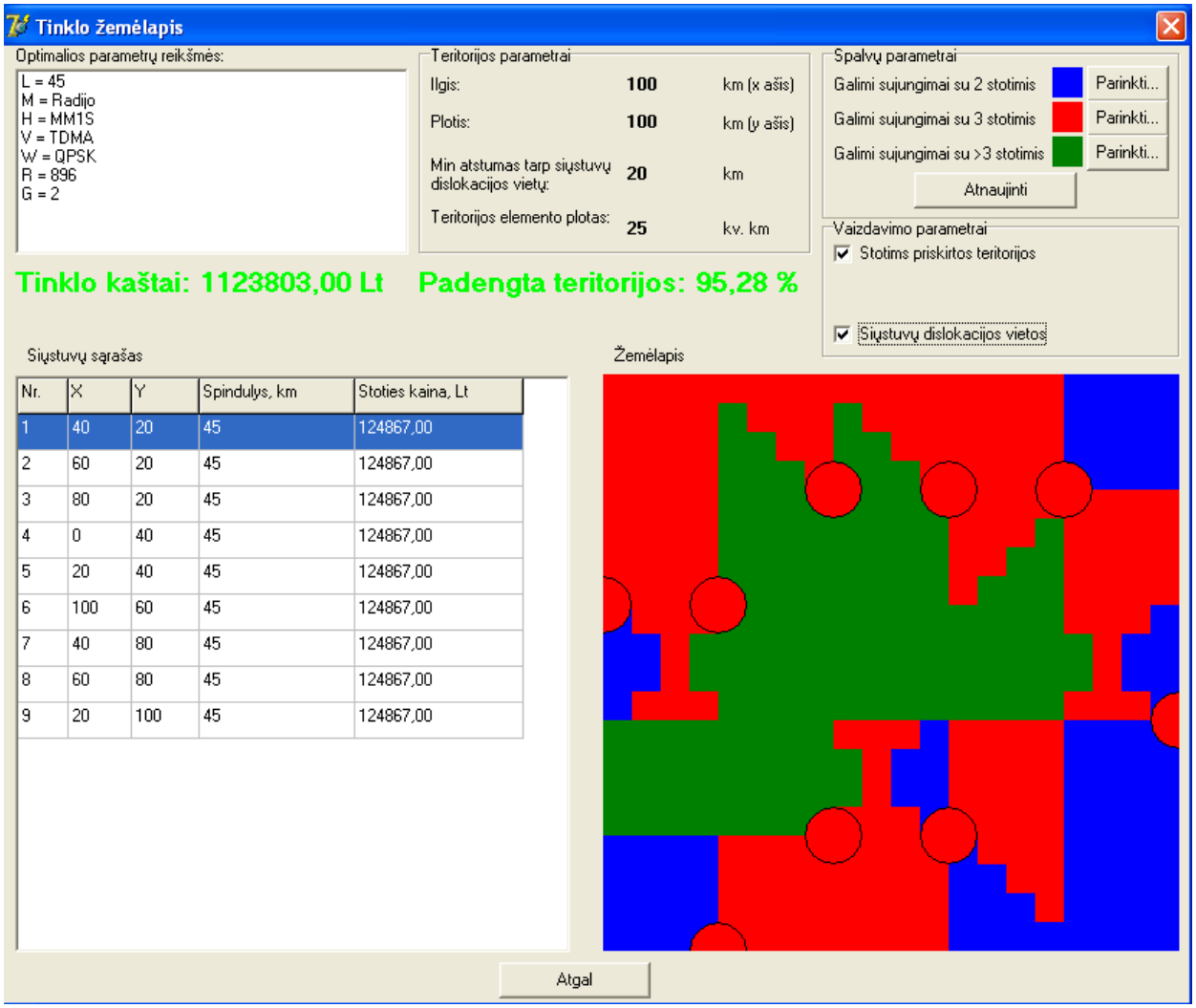
Šiame taške: $Q_1(X^*)$ = **29.628 s.** ir $Q_2(X^*)$ = **33.379,** taigi tenkinama apribojimų sistema:

$$\begin{cases} 0 \le Q_1(X^*) \le 30 \\ 3.54 \le Q_2(X^*) \le 100 \end{cases}$$

X^* parametrais nusakomo tinklo diegimo kaštai yra: $C_{min}= f(X^*)$ = **1123803,00 Lt.**

Teritorijos padengimo procentas – **95,28 %**

Programos rezultatų lango vaizdas:



Nr.	X	Y	Spindulys, km	Stoties kaina, Lt
1	40	20	45	124867,00
2	60	20	45	124867,00
3	80	20	45	124867,00
4	0	40	45	124867,00
5	20	40	45	124867,00
6	100	60	45	124867,00
7	40	80	45	124867,00
8	60	80	45	124867,00
9	20	100	45	124867,00

3.18 pav. Optimizavimo rezultatai fiksuoto telefono ryšio tinklui 3

Optimizavimo rezultatai mobiliojo telefono ryšio tinklui, kai stotims išdėstyti naudojamas modifikuotas „Skruzdžių kolonijos“ algoritmas, pateiktas 3.3.1 skyrelyje.

Stočių išdėstymo algoritmo parametrų reikšmės:

- Iteracijų skaičius = 5
- „Skruzdžių“ skaičius = 5

- $\beta = 0{,}4$
- $\rho = 0{,}5$
- Algoritmo kartojimų skaičius – 1 kartas

Gautas sprendinys $X^* =$ **{45; Radijo; MM1S, TDMA, 4QAM, 832, 1};**

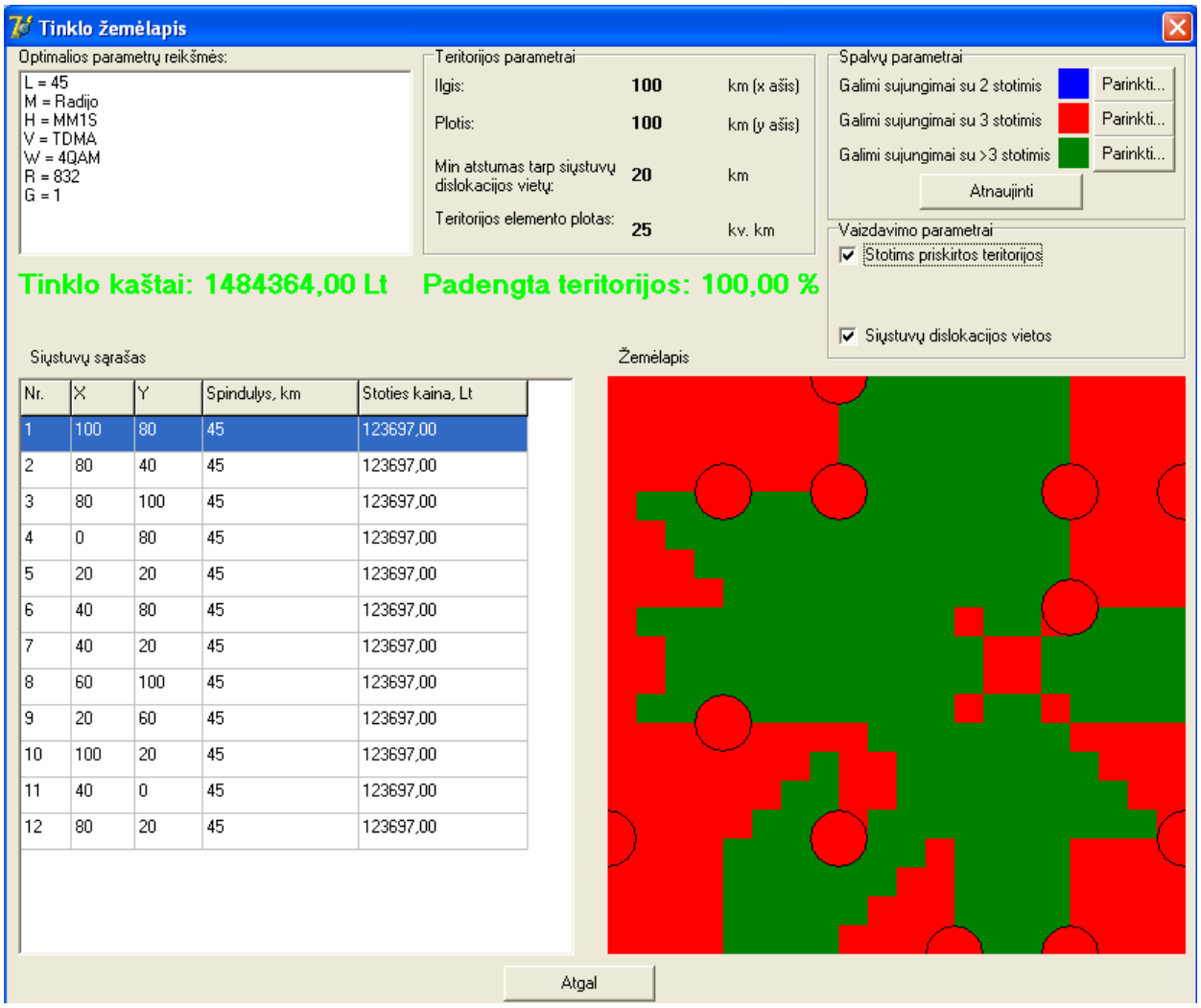
Šiame taške: $Q_1(X^*) =$ **29.513 s.** ir $Q_2(X^*) =$ **33.411,** taigi tenkinama apribojimų sistema:

$$\begin{cases} 0 \le Q_1(X^*) \le 30 \\ 3.54 \le Q_2(X^*) \le 100 \end{cases}$$

X^* parametrais nusakomo tinklo diegimo kaštai yra: $C_{min} = f(X^*) =$ **1484364,00 Lt.**

Teritorijos padengimo procentas – **100,00 %**

Programos rezultatų lango vaizdas:



Nr.	X	Y	Spindulys, km	Stoties kaina, Lt
1	100	80	45	123697,00
2	80	40	45	123697,00
3	80	100	45	123697,00
4	0	80	45	123697,00
5	20	20	45	123697,00
6	40	80	45	123697,00
7	40	20	45	123697,00
8	60	100	45	123697,00
9	20	60	45	123697,00
10	100	20	45	123697,00
11	40	0	45	123697,00
12	80	20	45	123697,00

3.19 pav. Optimizavimo rezultatai fiksuoto telefono ryšio tinklui 4

4. PROGRAMINĖ REALIZACIJA IR INSTRUKCIJA VARTOTOJUI

Realizuosime 3.2. skyrelyje aprašytą algoritmą programinio paketo „Borland Delphi 7“ aplinkoje. Ši aplinka buvo pasirinkta dėl jos teikiamų galimybių kuriant vartotojo sąsajos reikalaujančią programinę įrangą. Kituose skyreliuose bus nagrinėjama sukurtos programos galimybės ir jos veikimo principai. Programa gali būti naudojama bet kurioje iš šių operacinių sistemų:

- „Microsoft Windows 95/98/98SE/ME“
- „Microsoft Windows NT/2000/XP/Vista“

Pageidautina, kad kompiuteris, kuriame bus naudojama programa, turėtų kiek įmanoma galingesnį procesorių.

Programos struktūros aprašymas

Kaip matyti iš algoritmo aprašymo, jo rezultatas priklauso nuo šių veiksnių:

- Apribojimų skaičiaus ir jų struktūros;
- Tikslo funkcijos struktūros;
- Kintamųjų ir jų reikšmių;
- Stočių išdėstymo algoritmo.

Atsižvelgus į šiuos veiksnius, algoritmas realizuotas taip, kad programa būtų nepriklausoma nuo aukščiau išvardintų veiksnių. Tai yra realizuojama apribojimų, kaštų bei stočių išdėstymo skaičiavimus perkeliant į dinamines bibliotekas, kurių skaičius, veikimo principai ir kita yra neribojami.

Būtina sąlyga, keliama apribojimų ir kaštų moduliams, yra ta, kad jie eksportuotų šias funkcijas:

- Surikiuoti – vykdo kintamųjų ir jų reikšmių rikiavimą pagal apribojimą (kaštus)
- Apskaičiuoti – apskaičiuoja apribojimo (kaštų) reikšmę pagal kintamųjų reikšmes

Šių funkcijų pagalba vyksta komunikacija tarp kamieninės programos ir apribojimų bei kaštų modulių.

Pateikiame šių funkcijų detalius aprašus:

- Surikiuoti(***Kintamųjų sąrašas***): ***Kintamųjų ir jų reikšmių rikiuotė pagal apribojimą (kaštus);***
- Apskaičiuoti(***Kintamųjų sąrašas***): ***Pagal apribojimą (kaštus) apskaičiuota rodiklio reikšmė.***

Būtina sąlyga stočių išdėstymo moduliams yra ta, kad jie eksportuotų tokias funkcijas:

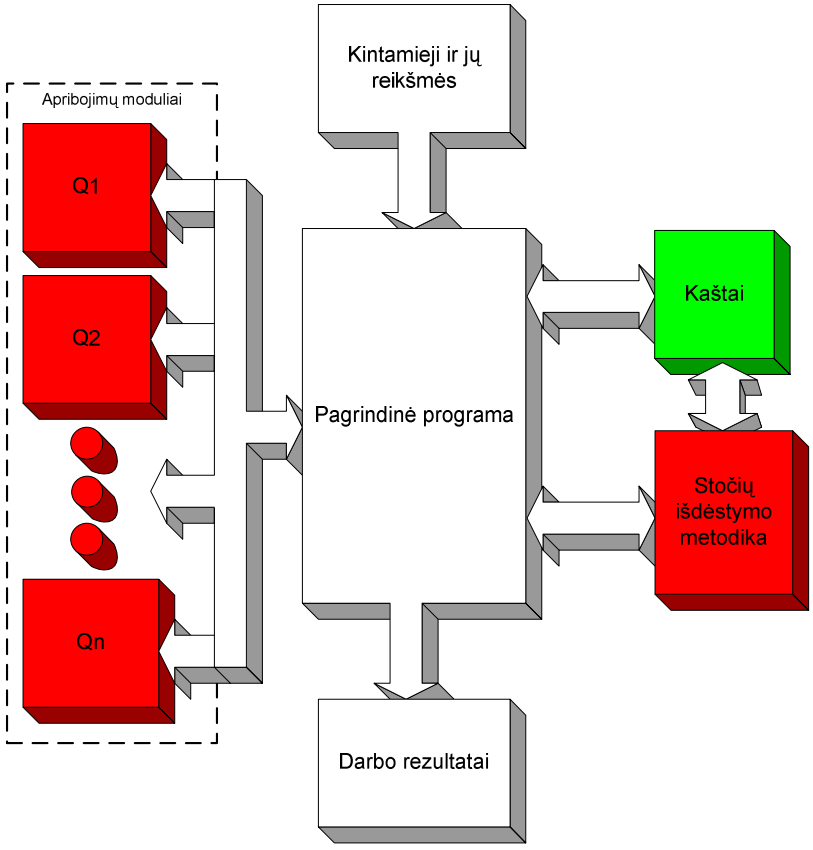
- Surikiuoti – vykdo kintamųjų ir jų reikšmių rikiavimą pagal įtaką maksimaliam teritorijos padengimo procentui;
- „Išdėstyti siųstuvus“ – iš pateiktų galimų stočių dislokacijos vietų sąrašo išrenka optimalias stočių dislokavimo koordinates.

Šių funkcijų pagalba vyksta komunikacija tarp kamieninės programos ir stočių išdėstymo modulio.

Pateikiame šių funkcijų detalius aprašus:

- Surikiuoti(***Kintamųjų sąrašas***): ***Kintamųjų ir jų reikšmių rikiuotė pagal apribojimą (kaštus);***
- Apskaičiuoti(***Kintamųjų sąrašas, Potencialių stočių dislokacijos vietų sąrašas, teritorijos žemėlapis, Stočių išdėstymo algoritmo parametrai***): ***Apskaičiuotos optimalios stočių dislokacijos vietos.***

Nubraižome duomenų srautų schemą programoje (4.1 pav.):



4.1 pav. Duomenų srautų schema programoje

Programos parametrai:

Kintamųjų reikšmės ir aprašai nuskaitomi iš šių bylų:

- „KintReiks.Txt“ – kintamųjų galimų reikšmių sąrašas;
- „Kintamieji.Txt“ – kintamųjų pavadinimų ir jų aprašų sąrašas.

Apribojimų reikšmės ir aprašymai nuskaitomi iš šių bylų:

- „Apribojimai.Txt“ – apribojimų pavadinimų ir jų aprašų sąrašas;
- „ApribDetal.Txt“ – apribojimų leistinų įgyti reikšmių sąrašas.

Kaštų metodikos pavadinimas ir aprašas nuskaitomas iš bylos „Kastai.txt“.

Stočių išdėstymo metodikų aprašai bei parametrai nuskaitomi iš šių bylų:

- „StociuIsdestymas.Txt“ – stočių metodikų pavadinimų ir jų aprašų sąrašas;
- „StociuIsdestymasDetal.Txt“ – stočių metodikų parametrų bei jų reikšmių sąrašas.

Stočių aprašai bei potencialios jų dislokacijos vietos nuskaitomos iš bylos arba generuojamos pagal vartotojo įvestus teritorijos parametrus: teritorijos ilgį, plotį, atstumus tarp dviejų siųstuvų ir pan.

Vartotojų pasiskirstymo žemėlapis nuskaitomas iš bylos arba generuojamas pagal vartotojo įvestus teritorijos parametrus atsitiktinai.

Visais atvejais vartotojas gali minėtus žemėlapius keisti programos veikimo metu taip, kad atitiktų jo poreikius.

Programos valdymo komandos

Pagrindinio lango valdymo komandos:

- „Parametrai bei jų reikšmės“ atidaro langą, skirtą optimizavimui naudojamų kintamųjų bei jų reikšmių parinkimui;
- „Kokybės kriterijai bei jų reikšmės“ atidaro langą, skirtą kokybės kriterijų bei leistinų jų reikšmių parinkimui;
- „Kaštų skaičiavimo metodika“ atidaro langą, skirtą kaštų metodikos parinkimui;
- „Stočių išdėstymo metodika“ atidaro langą, skirtą stočių išdėstymo algoritmo parinkimui ir jo parametrų nustatymui;
- „Teritorijos žemėlapis“ atidaro langą, skirtą teritorijos žemėlapio suformavimui;
- „Potencialių stočių dislokacijos vietų žemėlapis“ atidaro langą, skirtą potencialių stočių buvimo vietų parinkimui ir jų parametrų nustatymui;
- „Programos konfigūracija“ atidaro langą, skirtą algoritmo parametrų parinkimui ir prieigos tinklo struktūros nustatymui.

Programos rezultatų pateikimas

Viršutinėje lango dalyje vaizduojamos tinklo parametrų optimalios reikšmės, apskaičiuoti tinklo kaštai bei aptarnaujamos teritorijos procentas.

Apatinėje – pasirinktų stočių sąrašas su informacija apie stotims priskirtus žemėlapio tinklelio elementus. Skirtingos spalvos parodo skirtingoms stotims priskirtus tinklelio elementus. Raudoni apskritimai – stočių dislokacijos vietas. Viršutiniame dešiniajame lango kampe parodytos trys spalvos naudojamos pažymėti teritorijos tinklelio elementams, kuriuose įmanomi sujungimai su daugiau nei su viena stotimi. 3.16, 3.17, 3.18, 3.19 paveikslėliuose pateikiami programos rezultatų vaizdavimo langų pavyzdžiai.

5. DISKUSIJA

5.1. STOČIŲ IŠDĖSTYMO ALGORITMŲ PARAMETRŲ TYRIMŲ APŽVALGA

Mobiliojo telefono ryšio tinklo atvejis

5.1 lentelė

Genetinio algoritmo parametrų apžvalga mobiliojo telefono ryšio tinklo atveju

Populiacijos dydis	Iteracijų skaičius
3.14 pav. matyti, kad didinant parametro reikšmes didėja tikimybė rasti geresnį sprendinį, tačiau tuo pat metu smarkiai padidėja resursų poreikis skaičiavimams.	3.12 pav. matyti, kad didinant parametro reikšmes didėja tikimybė rasti geresnį sprendinį, tačiau tuo pat metu smarkiai padidėja resursų poreikis skaičiavimams.

Lyginant abiejų parametrų įtaką rezultatams galima pastebėti, kad populiacijos dydžio parametras yra svarbesnis nei iteracijų skaičius jei nenaudojama lokalios paieškos procedūra. Todėl esant ribotiems kompiuterio resursams tikslinga didinti populiacijos dydžio reikšmę iteracijų skaičiaus sąskaita. Naudojant lokalios paieškos procedūra, abiejų parametrų įtaka rezultatams panaši. Visų eksperimentų metu buvo pastebėtas faktas, kad lokalios paieškos procedūra sprendinio kokybę pagerino.

5.2 lentelė

„Skruzdžių kolonijos“ algoritmo parametrų apžvalga mobiliojo telefono ryšio tinklo atveju

„Skruzdžių“ skaičius	Iteracijų skaičius
3.8 pav. matyti, kad didinant parametro reikšmes didėja tikimybė rasti geresnį sprendinį, tačiau tuo pat metu smarkiai padidėja resursų poreikis skaičiavimams.	3.10 pav. matyti, kad didinant parametro reikšmes didėja tikimybė rasti geresnį sprendinį, tačiau tuo pat metu smarkiai padidėja resursų poreikis skaičiavimams.

Lyginant abiejų parametrų įtaką rezultatams galima pastebėti, kad taikant lokalios paieškos procedūrą parametrų įtaka yra mažesnė nei „gryno“ skruzdžių kolonijos algoritmo. Stebimas mažas iteracijų skaičiaus parametro įtakos pranašumas prieš „skruzdžių“ skaičiaus parametrą. Todėl esant ribotiems kompiuterio resursams tikslinga didinti iteracijų skaičiaus reikšmę „skruzdžių“ skaičiaus

sąskaita. Visų eksperimentų metu buvo pastebėtas faktas, kad lokalios paieškos procedūra sprendinio kokybę pagerino.

Fiksuoto telefono ryšio tinklo atvejis

5.3 lentelė

Genetinio algoritmo parametrų apžvalga fiksuoto telefono ryšio tinklo atveju

Populiacijos dydis	Iteracijų skaičius
3.13 pav. matyti, kad didinant parametro reikšmes didėja tikimybė rasti geresnį sprendinį, tačiau sprendinio kokybės pokytis yra mažesnis nei mobiliojo telefono ryšio tinklo atveju. Taipogi dėl skaičiavimų gausos resursų poreikis skaičiavimams padidėja labiau nei mobiliojo telefono ryšio tinklo atveju.	3.15 pav. matyti, kad didinant parametro reikšmes didėja tikimybė rasti geresnį sprendinį, tačiau sprendinio kokybės pokytis yra mažesnis nei mobiliojo telefono ryšio tinklo atveju. Taipogi dėl skaičiavimų gausos resursų poreikis skaičiavimams padidėja labiau nei mobiliojo telefono ryšio tinklo atveju.

Lyginant abiejų parametrų įtaką rezultatams galima pastebėti, kad parametrų reikšmių pokyčio įtaka algoritmo rezultatams nėra tokia didelė kaip mobiliojo telefono ryšio tinklo atveju. Nors abiejų parametrų pokyčio įtaka rezultatų pokyčiui yra panaši, didesnės populiacijos dydžio parametro reikšmės padeda rasti geresnius sprendinius, nei tai pavyktų padaryti didinant iteracijų skaičiaus parametro reikšmes. Šiuo atveju lokalios paieškos procedūra sprendinio kokybę pagerina labai ženkliai ir jos įtaka rezultatui yra didžiausia iš visų nagrinėtų variantų. Šis reiškinys leidžia modifikuotam genetiniam algoritmui konkuruoti su modifikuotu „skruzdžių kolonijos“ algoritmu, kai tuo tarpu „gryna“ genetinio algoritmo versija smarkiai nusileidžia „skruzdžių kolonijos“ algoritmo analogui.

5.4 lentelė

„Skruzdžių kolonijos“ algoritmo parametrų apžvalga fiksuoto telefono ryšio tinklo atveju

„Skruzdžių“ skaičius	Iteracijų skaičius
3.9 pav. matyti, kad didinant parametro reikšmes didėja tikimybė rasti geresnį sprendinį, tačiau tuo pat metu smarkiai padidėja resursų poreikis skaičiavimams.	3.11 pav. matyti, kad didinant parametro reikšmes didėja tikimybė rasti geresnį sprendinį, tačiau tuo pat metu smarkiai padidėja resursų poreikis skaičiavimams.

Lyginant abiejų parametrų įtaką rezultatams galima pastebėti, kad taikant lokalios paieškos procedūrą parametrų įtaka yra mažesnė nei „gryno“ „skruzdžių kolonijos“ algoritmo atveju. Lyginant su mobiliojo telefono ryšio tinklo atveju, stebimas ženklus iteracijų skaičiaus parametro įtakos pranašumas prieš „skruzdžių“ skaičiaus parametrą. Todėl esant ribotiems kompiuterio resursams tikslinga didinti iteracijų skaičiaus reikšmę „skruzdžių“ skaičiaus sąskaita. Visų eksperimentų metu buvo pastebėtas faktas, kad lokalios paieškos procedūra sprendinio kokybę pagerino.

Apibendrinant tai, kas parašyta, galima suformuluoti tokias išvadas:

- Jei leidžia turimi resursai, patartina rinktis kiek galima didesnės nagrinėtų parametrų reikšmes, nes didesnių reikšmių įtaka yra teigiama. „Skruzdžių kolonijos“ algoritmo atveju pirmenybę reikėtų teikti iteracijų skaičiaus parametro reikšmės didinimui, tuo tarpu genetinio algoritmo atveju pirmenybę reikėtų teikti populiacijos dydžio parametro reikšmės didinimui.
- Tikslinga taikyti abu algoritmus papildytus lokalios paieškos procedūromis, nes visais atvejais šių papildymų įtaka buvo tik teigiama.

5.2. ALGORITMŲ PRAKTINIO PANAUDOJIMO GALIMYBĖS

Tiek mobiliojo, tiek fiksuoto telefono ryšio tinklo atvejais norint gauti pilną teritorijos padengimą, tikslinga taikyti modifikuotą „skruzdžių kolonijos“ algoritmą, nes šiuo algoritmu leistinų sprendinių aibė apibrėžiama vienareikšmiškai: visi leistini sprendiniai garantuoja 100 % teritorijos padengimą. Modifikuotą genetinį algoritmą tikslinga naudoti, kai tenkina mažesnis teritorijos padengimo procentas. Naudodami 3.11 formule apibrėžiamą kokybės funkcijos išraišką, gauname sprendinius, kurių teritorijos padengimo procentas ~ 90 – 97 %, dėl to apskaičiuotos tinklo kaštų reikšmės mažesnės.

Vertinant 1 % teritorijos padengimo kaštus, pirmenybę reikėtų atiduoti modifikuotam genetiniui algoritmui, o sprendžiant problemas, kurios reikalauja 100 % teritorijos padengimo, pirmenybė tenka modifikuotam „skruzdžių kolonijos“ algoritmui.

Ypač tikslinga modifikuotą genetinį algoritmą taikyti tada, kai pagal duotas potencialias siųstuvų buvimo vietas 100 % teritorijos padengimo sukonstruoti neįmanoma. Tada modifikuotas genetinis algoritmas radęs išdėstymo variantą, apimantį < 100 %, bus smarkiai pranašesnis, nes modifikuotas „skruzdžių kolonijos“ algoritmas sprendinio nerastų.

Iš viso to, kas pasakyta, galima suformuoti šias naudojimo rekomendacijas:

- Jei yra galimybės ir poreikis konstruoti 100 % teritorijos aptarnavimą, tikslinga naudoti modifikuotą „skruzdžių kolonijos“ algoritmą;
- Jei galimybės sukonstruoti 100 % teritorijos aptarnavimą nėra arba ji nereikalinga, tikslinga naudoti modifikuotą genetinį algoritmą.

IŠVADOS

1) Jei yra galimybių ir poreikis aptarnauti 100 % teritorijos, tikslinga naudoti modifikuotą „skruzdžių kolonijos“ algoritmą.

2) Neturint galimybių arba poreikio aptarnauti 100 % teritorijos, tikslinga naudoti modifikuotą genetinį algoritmą.

3) Stočių išdėstymui naudojant modifikuotą „skruzdžių kolonijos“ algoritmą, tikslinga didinti iteracijų skaičiaus parametro ir „skruzdžių“ skaičiaus parametro reikšmes pirmenybę teikiant iteracijų skaičiaus parametro reikšmės didinimui.

4) Stočių išdėstymui naudojant modifikuotą genetinį algoritmą, tikslinga didinti iteracijų skaičiaus parametro ir populiacijos dydžio parametro reikšmes pirmenybę teikiant populiacijos dydžio parametro reikšmės didinimui.

5) Pateikti stočių išdėstymo algoritmai turi šiuos privalumus:

- gali būti naudojami ir su kitais prieigos tinklo parametrų nustatymo algoritmais;
- gali būti naudojami nepriklausomai nuo daugiaparametrinio prieigos tinklo optimizavimo algoritmo, kai stočių parametrai yra fiksuoti ir nebūtinai vienodi visiems modeliuojamo tinklo segmentams;
- modifikavus individo kokybės įvertinimo funkciją genetiniam algoritmui ir tikimybės skaičiavimo funkciją „skruzdžių kolonijos“ algoritmui pateiktus algoritmus galima naudoti ir kitų teritorijos padengimo problemų sprendimui.

6) Modelyje prieigos tinklo kokybę charakterizuojantys rodikliai bei stočių išdėstymo metodikos gali būti pasirenkami laisvai, o tai leidžia optimizuoti skirtingos paskirties ir struktūros prieigos tinklus.

7) Pateiktas daugiaparametrinis prieigos tinklo optimizavimo algoritmas gali būti naudojamas ir kitų optimizavimo uždavinių su apribojimais sprendimui, kurių kintamųjų reikšmių aibės yra diskrečios ir sutvarkomos atskirai pagal įtaką tikslo ir apribojimų funkcijoms.

REKOMENDACIJOS

Tolimesnis parametrų bei jų reikšmių rikiuotėmis besiremiančio daugiaparametrinio prieigos tinklo optimizavimo algoritmo tobulinimas netikslingas, nes pagerinti sprendinių kokybę vargu ar pavyks. Būtų galima kurti naujus algoritmus, kurių duomenų apibrėžimo sritis yra platesnė. Vertėtų atkreipti dėmesį į genetinį ir modeliuojamo atkaitinimo algoritmus bei jų modifikacijas.

Aptartus stočių išdėstymo algoritmus galima nagrinėti toliau. Ypač tai pasakytina apie fiksuoto telefono ryšio atvejui skirtą genetinio algoritmo modifikaciją. Tikslinga būtų patyrinėti skirtingas šio modifikavimo algoritmo evoliucionavimo schemas. Taipogi galima pamėginti patobulinti lokalios paieškos procedūras taip, kad kiek galima mažiau nukentėtų jungtinių algoritmų konvergavimo greitis.

Šiame darbe pristatytą daugiaparametrinį prieigos tinklo optimizavimo algoritmo variantą galima naudoti kartu su kitais stočių išdėstymo algoritmais.

PADĖKOS

Dėkoju magistro darbo vadovui doc. dr. N. Listopadskiui už pagalbą ir patarimus. Taip pat dėkoju dr. V. Grimailai už pateiktus duomenis ir konsultacijas telekomunikacijų inžinerijos klausimais, FMMM-5 grupės magistrantams Juliui Simonavičiui ir Jonui Pokštui už naudingas diskusijas kombinatorinio optimizavimo temomis.

LITERATŪRA

1. P. Calégari, F. Guidec, P. Kuonen, B. Chamaret, S. Ubéda, S. Josselin, D. Wagner, M. Pizarosso. Combinatorial Optimization Algorithms, Theoretical Computer Science 265 (2000 m.) 1-2

2. Karen I. Aardal, Stan P. M. Van Hoesel, Arie M. C. A. Koster, Carlo Mannino, Antonio Sassano. Models and Solution Techniques for Frequency Assignment Problems. Quarterly journal of the Belgian, French and Italian operations research societies, 1(4), (2003 m.) 261-317

3. L. Young, P. Yum. Fixed Channel Assignment Optimization for Cellular Mobile Networks. Asia-Pacific Conference on Communications, (1995 m.) 573-577

4. Bernard Fortz. Combinatorial Optimization and Telecommunications. Handouts of the doctoral course given at CORE in September. 2003 m.

5. Carlos A.S. Oliveira, Panos M. Pardalos. A Survey of Combinatorial Optimization Problems in Multicast Routing. Computers & OR 32, (2005 m.) 1953-1981

6. Grimaila, Vitas. Telekomunikacijų prieigos tinklo daugiaparametrinis įvertinimas [Rankraštis]: daktaro disertacija: technologijos mokslai, elektros ir elektronikos inžinerija (01 T) / Vitas Grimaila; Kauno Technologijos Universitetas. – Kaunas. 2003 m. – 146 lap.: iliustr., žml.

7. MARCO DORIGO, THOMAS STUTZLE. The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. Kluwer Academic Publishers, (2002 m.)

8. P. Calégari, F. Guidec, P. Kuonen. Combinatorial Optimization Algorithms For Radio Network Planning. Theoretical Computer Science (TCS), Special Issue on Combinatorics and Computer Science, 265(1), (2001 m.) 235-245

9. Lazaravičius S., Listopadskis N. Teritorijos padengimo uždavinio analizė. VI studentų konferencija, KTU, Kaunas (2006 m.)

10. Lazaravičius S., Listopadskis N. Stočių išdėstymo algoritmo analizė. Lietuvos matematikų draugijos XLVII konferencija, Kaunas, (2006 m.)

11. Lazaravičius S., Listopadskis N. Prieigos tinklo parametrų optimalaus parinkimo algoritmas. Matematika ir matematikos dėstymas 2006, Kaunas, (2006 m.)

1 PRIEDAS. KINTAMŲJŲ, KOKYBĖS RODIKLIŲ BEI KAŠTŲ FUNKCIJOS REIKŠMIŲ LENTELĖS

1.1 lentelė

Bylos „Kintamieji.Txt“ pavyzdys

L	Aprėpties zonos spindulys (km)
M	Kanalo / Perdavimo terpes tipas
H	Duomenų paketu aptarnavimo disciplina
V	Prieigos prie kanalo metodas
W	Moduliacijos tipas
R	Duomenų perdavimo sparta (kbps)
G	Kodeko tipas

1.2 lentelė

Bylos „Apribojimai.Txt“ pavyzdys

Q1	Duomenų paketu vėlinimas
Q2	Signalo ir trukdžio santykis

1.3 lentelė

Bylos „KintReiks.Txt“ pavyzdys

L	10
50	
45	
40	
35	
30	
25	
20	
15	
10	
5	
M	**1**
Radijo	
H	**1**
MM1S	
V	**1**
TDMA	
W	**2**
QPSK	
4QAM	
R	**16**
64	

128
192
256
320
384
448
512
576
640
704
768
832
896
960
1024

G	**5**
1	
2	
30	
60	
82	

1.4 lentelė

Bylos ,,ApribDetal.Txt“ pavyzdys

Q1	
0	30
Q2	
3.54	100

1.5 lentelė

Q_1 = f(R, G) reikšmių pavyzdys

	G1	**G2**	**G3**	**G4**	**G5**
R1	1001.15	1002.15	1030.15	1060.15	1082.15
R2	177.96	178.96	206.96	236.96	258.96
R3	94.52	95.52	123.52	153.52	175.52
R4	67.28	68.28	96.28	126.28	148.28
R5	54.13	55.13	83.13	113.13	135.13
R6	46.47	47.47	75.47	105.47	127.47
R7	41.48	42.48	70.48	100.48	122.48
R8	37.98	38.98	66.98	96.98	118.98
R9	35.39	36.39	64.39	94.39	116.39
R10	33.41	34.41	62.41	92.41	114.41
R11	31.84	32.84	60.84	90.84	112.84
R12	30.57	31.57	59.57	89.57	111.57
R13	29.51	30.51	58.51	88.51	110.51
R14	28.63	29.63	57.63	87.63	109.63
R15	27.87	28.87	56.87	86.87	108.87
R16	27.22	28.22	56.22	86.22	108.22

1.6 lentelė

$Q_2 = f(L, R)$ reikšmių pavyzdys

R\L	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	L15	L16	L17	L18	L19	L20
R1	43.65	36.33	32.05	29.02	26.66	24.74	23.11	21.70	20.46	19.35	18.34	17.42	16.58	15.80	15.07	14.39	13.75	13.14	12.57	12.03
R2	41.54	34.22	29.95	26.91	24.55	22.63	21.00	19.59	18.35	17.24	16.23	15.32	14.47	13.69	12.96	12.28	11.64	11.04	10.47	9.92
R3	41.36	34.05	29.77	26.73	24.38	22.45	20.83	19.42	18.18	17.06	16.06	15.14	14.29	13.51	12.78	12.10	11.46	10.86	10.29	9.75
R4	41.24	33.92	29.64	26.61	24.25	22.33	20.70	19.29	18.05	16.94	15.93	15.01	14.17	13.39	12.66	11.98	11.34	10.74	10.16	9.62
R5	41.14	33.83	29.55	26.51	24.16	22.23	20.61	19.20	17.95	16.84	15.84	14.92	14.07	13.29	12.56	11.88	11.24	10.64	10.07	9.53
R6	41.06	33.75	29.47	26.43	24.08	22.15	20.53	19.12	17.87	16.76	15.76	14.84	13.99	13.21	12.48	11.80	11.16	10.56	9.99	9.45
R7	41.00	33.68	29.40	26.37	24.01	22.09	20.46	19.05	17.81	16.70	15.69	14.77	13.93	13.14	12.42	11.74	11.10	10.49	9.92	9.38
R8	40.94	33.62	29.34	26.31	23.95	22.03	20.40	18.99	17.75	16.64	15.63	14.71	13.87	13.09	12.36	11.68	11.04	10.43	9.86	9.32
R9	40.89	33.57	29.29	26.26	23.90	21.98	20.35	18.94	17.70	16.59	15.58	14.66	13.82	13.04	12.31	11.63	10.99	10.38	9.81	9.27
R10	40.84	33.53	29.25	26.21	23.86	21.93	20.30	18.90	17.65	16.54	15.53	14.62	13.77	12.99	12.26	11.58	10.94	10.34	9.77	9.23
R11	40.80	33.48	29.20	26.17	23.81	21.89	20.26	18.85	17.61	16.50	15.49	14.57	13.73	12.95	12.22	11.54	10.90	10.30	9.73	9.18
R12	40.76	33.45	29.17	26.13	23.78	21.85	20.23	18.82	17.57	16.46	15.46	14.54	13.69	12.91	12.18	11.50	10.86	10.26	9.69	9.15
R13	40.73	33.41	29.13	26.10	23.74	21.82	20.19	18.78	17.54	16.43	15.42	14.50	13.66	12.88	12.15	11.47	10.83	10.22	9.65	9.11
R14	40.69	33.38	29.10	26.06	23.71	21.79	20.16	18.75	17.51	16.39	15.39	14.47	13.63	12.84	12.12	11.43	10.79	10.19	9.62	9.08
R15	40.66	33.35	29.07	26.03	23.68	21.76	20.13	18.72	17.48	16.36	15.36	14.44	13.60	12.81	12.09	11.40	10.76	10.16	9.59	9.05
R16	40.64	33.32	29.04	26.01	23.65	21.73	20.10	18.69	17.45	16.34	15.33	14.41	13.57	12.79	12.06	11.38	10.74	10.13	9.56	9.02

1.7 lentelė

$C = f(L, R)$ reikšmių pavyzdys

R\L	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	L15	L16	L17	L18	L19	L20
R1	64520.6	16667	7777	4678.2	3255.8	2443	1985.8	1681	1477.8	1325.4	1223.8	1122.2	1071.4	1020.6	969.8	919	919	868.2	868.2	868.2
R2	64620.6	16767	7877	4778.2	3355.8	2543	2085.8	1781	1577.8	1425.4	1323.8	1222.2	1171.4	1120.6	1069.8	1019	1019	968.2	968.2	968.2
R3	64800.6	16947	8057	4958.2	3535.8	2723	2265.8	1961	1757.8	1605.4	1503.8	1402.2	1351.4	1300.6	1249.8	1199	1199	1148.2	1148.2	1148.2
R4	65070.6	17217	8327	5228.2	3805.8	2993	2535.8	2231	2027.8	1875.4	1773.8	1672.2	1621.4	1570.6	1519.8	1469	1469	1418.2	1418.2	1418.2
R5	65430.6	17577	8687	5588.2	4165.8	3353	2895.8	2591	2387.8	2235.4	2133.8	2032.2	1981.4	1930.6	1879.8	1829	1829	1778.2	1778.2	1778.2
R6	65880.6	18027	9137	6038.2	4615.8	3803	3345.8	3041	2837.8	2685.4	2583.8	2482.2	2431.4	2380.6	2329.8	2279	2279	2228.2	2228.2	2228.2
R7	66420.6	18567	9677	6578.2	5155.8	4343	3885.8	3581	3377.8	3225.4	3123.8	3022.2	2971.4	2920.6	2869.8	2819	2819	2768.2	2768.2	2768.2
R8	67050.6	19197	10307	7208.2	5785.8	4973	4515.8	4211	4007.8	3855.4	3753.8	3652.2	3601.4	3550.6	3499.8	3449	3449	3398.2	3398.2	3398.2
R9	67770.6	19917	11027	7928.2	6505.8	5693	5235.8	4931	4727.8	4575.4	4473.8	4372.2	4321.4	4270.6	4219.8	4169	4169	4118.2	4118.2	4118.2
R10	68580.6	20727	11837	8738.2	7315.8	6503	6045.8	5741	5537.8	5385.4	5283.8	5182.2	5131.4	5080.6	5029.8	4979	4979	4928.2	4928.2	4928.2
R11	69480.6	21627	12737	9638.2	8215.8	7403	6945.8	6641	6437.8	6285.4	6183.8	6082.2	6031.4	5980.6	5929.8	5879	5879	5828.2	5828.2	5828.2
R12	70470.6	22617	13727	10628.2	9205.8	8393	7935.8	7631	7427.8	7275.4	7173.8	7072.2	7021.4	6970.6	6919.8	6869	6869	6818.2	6818.2	6818.2
R13	71550.6	23697	14807	11708.2	10285.8	9473	9015.8	8711	8507.8	8355.4	8253.8	8152.2	8101.4	8050.6	7999.8	7949	7949	7898.2	7898.2	7898.2
R14	72720.6	24867	15977	12878.2	11455.8	10643	10185.8	9881	9677.8	9525.4	9423.8	9322.2	9271.4	9220.6	9169.8	9119	9119	9068.2	9068.2	9068.2
R15	73980.6	26127	17237	14138.2	12715.8	11903	11445.8	11141	10937.8	10785.4	10683.8	10582.2	10531.4	10480.6	10429.8	10379	10379	10328.2	10328.2	0328.2
R16	75330.6	27477	18587	15488.2	14065.8	13253	12795.8	12491	12287.8	12135.4	12033.8	11932.2	11881.4	11830.6	11779.8	11729	11729	11678.2	11678.2	1678.2

2 PRIEDAS. DAUGIAPARAMETRINIO PRIEIGOS TINKLO OPTIMIZAVIMO ALGORITMO REZULTATAI

Optimizavimo kintamųjų, apribojimų bei kaštų funkcijos reikšmės po kiekvienos iteracijos fiksuoto telefono ryšio tinklui, kai stotims išdėstyti naudojamas modifikuotas Genetinis algoritmas, pateiktas 3.3.2 skyrelyje

2.1 lentelė

Bylos „Algoritmas.csv“ informacija atlikus skaičiavimus

	L	M	H	V	W	R	G	Q1	Q2	Kaštai
Leistino sprendinio paieška	50	Radijo	MM1S	TDMA	QPSK	320	1	**54,131**	41,141	
	50	Radijo	MM1S	TDMA	QPSK	384	1	**46,468**	41,062	
	50	Radijo	MM1S	TDMA	QPSK	448	1	**41,477**	40,995	
	50	Radijo	MM1S	TDMA	QPSK	512	1	**37,979**	40,937	
	50	Radijo	MM1S	TDMA	QPSK	576	1	**35,394**	40,886	
	50	Radijo	MM1S	TDMA	QPSK	640	1	**33,41**	40,84	
	50	Radijo	MM1S	TDMA	QPSK	704	1	**31,839**	40,799	
	50	Radijo	MM1S	TDMA	QPSK	768	1	**30,565**	40,761	
	50	Radijo	MM1S	TDMA	QPSK	832	1	**29,513**	40,726	
Optimalaus sprendinio paieška	50	Radijo	MM1S	TDMA	QPSK	832	1	29,513	40,726	**1196451**
	45	Radijo	MM1S	TDMA	QPSK	832	1	29,513	33,411	
	45	Radijo	MM1S	TDMA	QPSK	832	1	29,513	33,411	**963992,9**
	40	Radijo	MM1S	TDMA	QPSK	832	1	29,513	29,132	
	40	**Radijo**	**MM1S**	**TDMA**	**QPSK**	**832**	**1**	**29,513**	**29,132**	**938462,75**

Gautas sprendinys X^* = **{40; Radijo; MM1S, TDMA, QPSK, 832, 1};**

Optimizavimo kintamųjų, apribojimų bei kaštų funkcijos reikšmės po kiekvienos iteracijos fiksuoto telefono ryšio tinklui, kai stotims išdėstyti naudojamas modifikuotas „Skruzdžių kolonijos“ algoritmas, pateiktas 3.3.2 skyrelyje

2.2 lentelė

Bylos „Algoritmas.csv“ informacija atlikus skaičiavimus

	L	M	H	V	W	R	G	Q1	Q2	Kaštai
Leistino sprendinio paieška	50	Radijo	MM1S	TDMA	4QAM	768	1	**30,565**	40,761	
	50	Radijo	MM1S	TDMA	4QAM	832	1	**29,513**	40,726	

Optimalaus sprendinio paieška	50	Radijo	MM1S	TDMA	4QAM	832	1	29,513	40,726	**1208399,84**
	45	Radijo	MM1S	TDMA	4QAM	832	1	29,513	33,411	
	45	**Radijo**	**MM1S**	**TDMA**	**4QAM**	**832**	**1**	**29,513**	**33,411**	**984738,25**

Gautas sprendinys X^* = **{45; Radijo; MM1S, TDMA, 4QAM, 832, 1};**

Optimizavimo kintamųjų, apribojimų bei kaštų funkcijos reikšmės po kiekvienos iteracijos mobiliojo telefono ryšio tinklui, kai stotims išdėstyti naudojamas modifikuotas Genetinis algoritmas, pateiktas 3.3.1 skyrelyje

2.3 lentelė

Bylos „Algoritmas.csv“ informacija atlikus skaičiavimus

	L	M	H	V	W	R	G	Q1	Q2	Kaštai
Leistino sprendinio paieška	50	Radijo	MM1S	TDMA	QPSK	192	60	**153,519**	41,363	
	50	Radijo	MM1S	TDMA	QPSK	256	60	**126,281**	41,238	
	50	Radijo	MM1S	TDMA	QPSK	320	60	**113,131**	41,141	
	50	Radijo	MM1S	TDMA	QPSK	384	60	**105,468**	41,062	
	50	Radijo	MM1S	TDMA	QPSK	448	60	**100,477**	40,995	
	50	Radijo	MM1S	TDMA	QPSK	512	60	**96,979**	40,937	
	50	Radijo	MM1S	TDMA	QPSK	576	60	**94,394**	40,886	
	50	Radijo	MM1S	TDMA	QPSK	640	60	**92,41**	40,84	
	50	Radijo	MM1S	TDMA	QPSK	704	60	**90,839**	40,799	
	50	Radijo	MM1S	TDMA	QPSK	768	60	**89,565**	40,761	
	50	Radijo	MM1S	TDMA	QPSK	832	60	**88,513**	40,726	
	50	Radijo	MM1S	TDMA	QPSK	896	60	**87,628**	40,694	
	50	Radijo	MM1S	TDMA	QPSK	960	60	**86,874**	40,664	
	50	Radijo	MM1S	TDMA	QPSK	1024	60	**86,224**	40,636	
	50	Radijo	MM1S	TDMA	QPSK	1024	30	**56,224**	40,636	
	50	Radijo	MM1S	TDMA	QPSK	1024	2	**28,224**	40,636	
Optimalaus sprendinio paieška	50	Radijo	MM1S	TDMA	QPSK	1024	2	28,224	40,636	**1402645**
	45	Radijo	MM1S	TDMA	QPSK	1024	2	28,224	33,321	
	45	Radijo	MM1S	TDMA	QPSK	1024	2	28,224	33,321	**1147293**
	40	Radijo	MM1S	TDMA	QPSK	1024	2	28,224	29,042	
	45	Radijo	MM1S	TDMA	QPSK	960	2	28,874	33,349	
	45	Radijo	MM1S	TDMA	QPSK	960	2	28,874	33,349	**1135143**
	45	Radijo	MM1S	TDMA	QPSK	896	2	29,628	33,379	
	45	**Radijo**	**MM1S**	**TDMA**	**QPSK**	**896**	**2**	**29,628**	**33,379**	**1123803**

Gautas sprendinys X^* = **{45; Radijo; MM1S, TDMA, QPSK, 896, 2};**

Optimizavimo kintamųjų, apribojimų bei kaštų funkcijos reikšmės po kiekvienos iteracijos mobiliojo telefono ryšio tinklui, kai stotims išdėstyti naudojamas modifikuotas „Skruzdžių kolonijos“ algoritmas, pateiktas 3.3.1 skyrelyje

2.4 lentelė

Bylos „Algoritmas.csv“ informacija atlikus skaičiavimus

	L	M	H	V	W	R	G	Q1	Q2	Kaštai
Leistino sprendinio paieška	50	Radijo	MM1S	TDMA	4QAM	256	60	**126,281**	41,238	
	50	Radijo	MM1S	TDMA	4QAM	320	60	**113,131**	41,141	
	50	Radijo	MM1S	TDMA	4QAM	384	60	**105,468**	41,062	
	50	Radijo	MM1S	TDMA	4QAM	448	60	**100,477**	40,995	
	50	Radijo	MM1S	TDMA	4QAM	512	60	**96,979**	40,937	
	50	Radijo	MM1S	TDMA	4QAM	576	60	**94,394**	40,886	
	50	Radijo	MM1S	TDMA	4QAM	640	60	**92,41**	40,84	
	50	Radijo	MM1S	TDMA	4QAM	704	60	**90,839**	40,799	
	50	Radijo	MM1S	TDMA	4QAM	768	60	**89,565**	40,761	
	50	Radijo	MM1S	TDMA	4QAM	832	60	**88,513**	40,726	
	50	Radijo	MM1S	TDMA	4QAM	896	60	**87,628**	40,694	
	50	Radijo	MM1S	TDMA	4QAM	960	60	**86,874**	40,664	
	50	Radijo	MM1S	TDMA	4QAM	1024	60	**86,224**	40,636	
	50	Radijo	MM1S	TDMA	4QAM	1024	30	**56,224**	40,636	
	50	Radijo	MM1S	TDMA	4QAM	1024	2	**28,224**	40,636	
Optimalaus sprendinio paieška	50	Radijo	MM1S	TDMA	4QAM	1024	2	28,224	40,636	**2103967**
	45	Radijo	MM1S	TDMA	4QAM	1024	2	28,224	33,321	
	45	Radijo	MM1S	TDMA	4QAM	1024	2	28,224	33,321	**1529724**
	40	Radijo	MM1S	TDMA	4QAM	1024	2	28,224	29,042	
	45	Radijo	MM1S	TDMA	4QAM	960	2	28,874	33,349	
	45	Radijo	MM1S	TDMA	4QAM	960	2	28,874	33,349	**1513524**
	45	Radijo	MM1S	TDMA	4QAM	896	2	29,628	33,379	
	45	Radijo	MM1S	TDMA	4QAM	896	2	29,628	33,379	**1498404**
	45	Radijo	MM1S	TDMA	4QAM	832	2	30,513	33,411	
	45	Radijo	MM1S	TDMA	4QAM	832	1	29,513	33,411	
	45	**Radijo**	**MM1S**	**TDMA**	**4QAM**	**832**	**1**	**29,513**	**33,411**	**1484364**

Gautas sprendinys X^* = **{45; Radijo; MM1S, TDMA, 4QAM, 832, 1};**

3 PRIEDAS. PROGRAMOS TEKSTAI

unit BevielesPrieigosAlgoritmas; //Genetinis algoritmas

```
interface

uses
  Classes, SysUtils, Zonos, Siustuvai, Math, StociuMetodikosInfo, PadengimoVektorius,
  StdCtrls, DuomenuStrukturos;

type
  TBevielesPrieigosAlgoritmas = class
  private
    Populiacija: TPopuliacija;
    Siustuvai: TSiustuvaiList;
    Logas: TStrings;
    procedure GeneruotiGena(Genas: TGenas);
    procedure GeneruotiGenus(Chromosoma: TChromosoma);
    procedure GeneruotiPradinePopuliacija(Dydis: Integer);
    procedure IsrikiuotiPopuliacija(naujaPopuliacija: TPopuliacija);
    procedure DvitaskisKryzminimas(naujaPopuliacija: TPopuliacija);
    procedure LokaliPaieska(Nuo: Integer; naujaPopuliacija: TPopuliacija);
    procedure LokaliPaieskaChromosomai(Chromosoma: TChromosoma);
    procedure ElitoKopijavimas(naujaPopuliacija: TPopuliacija; ElitoKoef: Real);
    procedure Mutavimas(naujaPopuliacija: TPopuliacija; MutavimoKoef: Real);
    function PadengimoProcentas(Chromosoma: TChromosoma): Real;
    function ArGalimaPasalinti(num: Integer; Chromosoma: TChromosoma): boolean;
    function AtrinktiTeva(Tevas: Integer): TChromosoma;
    function AtnaujintiPopuliacija(PopuliacijosDydis: Integer;
                                   MutavimoKoef: Real;
                                   ElitoKoef: Real;
                                   NaudojamaLokaliPaieska: boolean): boolean;
    function ChromosomosKokybe(Chromosoma: TChromosoma): Real;
    function GenetinisAlgoritmas(IteracijuSk, PopuliacijosDydis: Integer;
                                 NaudojamaLokaliPaieska: boolean;
                                 MutavimoKoef: Real; ElitoKoef: Real;
                                 out maxKokybesChromosoma: TChromosoma): boolean;
    function TinkloKastai(Sprendinys: TSiustuvaiList): real;
  protected

  public
    MinLeistinasPadengimas: Real;
    function Apskaiciuoti(Siustuvai1: TSiustuvaiList; Parametrai: TMetodikosParamList;
                          Log: TStrings;
                          out Sprendinys: TSiustuvaiList; out Kaina, Padengta: real): boolean;
  end;

implementation

function TBevielesPrieigosAlgoritmas.ArGalimaPasalinti(num: Integer;
                                                       Chromosoma: TChromosoma): boolean;
var suminisPadengimas: TPadengimoVektorius;
    i, j: Integer;
begin
  suminisPadengimas:= TPadengimoVektorius.Create;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    suminisPadengimas.Add;
  for i:= 0 to Chromosoma.Genai.Count - 1 do
    if Chromosoma.Genai[i].Reiksme > 0 then
      if i <> num then
        for j:= 0 to Siustuvai[i].PadengimoVektorius.Count - 1 do
          suminisPadengimas[j].Padengta:= suminisPadengimas[j].Padengta +
                                          Siustuvai[i].PadengimoVektorius[j].Padengta;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    if suminisPadengimas[i].Padengta < Siustuvai.Zonos[i].PadengimoLaipsnis then
    begin
      Result:= false;
      suminisPadengimas.Free;
      suminisPadengimas:= nil;
      Exit;
    end;
  Result:= true;
  suminisPadengimas.Free;
```

```
  suminisPadengimas:= nil;
end;

procedure TBevielesPrieigosAlgoritmas.LokaliPaieskaChromosomai(Chromosoma: TChromosoma);
var i: Integer;
begin
  for i:= 0 to Chromosoma.Genai.Count - 1 do
    if Chromosoma.Genai[i].Reiksme > 0 then
      if ArGalimaPasalinti(i, Chromosoma) then
        Chromosoma.Genai[i].Reiksme:= 0;
end;

procedure TBevielesPrieigosAlgoritmas.LokaliPaieska(Nuo: Integer; naujaPopuliacija: TPopuliacija);
var i: Integer;
begin
  for i:= Nuo to naujaPopuliacija.Count - 1 do
    LokaliPaieskaChromosomai(naujaPopuliacija[i]);
end;

procedure TBevielesPrieigosAlgoritmas.GeneruotiGena(Genas: TGenas);
begin
  if Random >= 0.5 then
    Genas.Reiksme:= 1;
end;

procedure TBevielesPrieigosAlgoritmas.GeneruotiGenus(Chromosoma: TChromosoma);
var i: Integer;
    oldProc, proc: real;
begin
  Chromosoma.Genai.Clear;
  while Chromosoma.Genai.Count < Siustuvai.Count do
    Chromosoma.Genai.Add;
  Randomize;
  repeat
    oldProc := PadengimoProcentas(Chromosoma);
    for i := 0 to Chromosoma.Genai.Count - 1 do
      if Chromosoma.Genai[i].Reiksme = 0 then
        GeneruotiGena(Chromosoma.Genai[i]);
    proc := PadengimoProcentas(Chromosoma);
  until (proc >= MinLeistinasPadengimas) or (oldProc >= proc);
end;

procedure TBevielesPrieigosAlgoritmas.GeneruotiPradinePopuliacija(Dydis: Integer);
begin
  if Populiacija = nil then
    Populiacija:= TPopuliacija.Create
  else
    Populiacija.Clear;
  while Populiacija.Count < Dydis do
    GeneruotiGenus(Populiacija.Add);
end;

function TBevielesPrieigosAlgoritmas.PadengimoProcentas(Chromosoma: TChromosoma): Real;
var i, j, visoElementu, padengtaElementu: Integer;
    suminisPadengimas: TPadengimoVektorius;
begin
  suminisPadengimas:= TPadengimoVektorius.Create;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    suminisPadengimas.Add;
  for i:= 0 to Chromosoma.Genai.Count - 1 do
    if Chromosoma.Genai[i].Reiksme > 0 then
      for j:= 0 to Siustuvai.Zonos.Count - 1 do
        suminisPadengimas[j].Padengta:= suminisPadengimas[j].Padengta +
                                        Siustuvai[i].PadengimoVektorius[j].Padengta;
  visoElementu:= 0;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    visoElementu:= visoElementu + Siustuvai.Zonos[i].PadengimoLaipsnis;
  padengtaElementu:= 0;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    if suminisPadengimas[i].Padengta < Siustuvai.Zonos[i].PadengimoLaipsnis then
      padengtaElementu:= padengtaElementu + suminisPadengimas[i].Padengta
    else
      padengtaElementu:= padengtaElementu + Siustuvai.Zonos[i].PadengimoLaipsnis;
  if visoElementu > 0 then
    Result:= padengtaElementu / visoElementu
  else
    Result:= 1;
  suminisPadengimas.Free;
  suminisPadengimas:= nil;
end;
```

```
function TBevielesPrieigosAlgoritmas.ChromosomosKokybe(Chromosoma: TChromosoma): Real;
var i, j, visoElementu, padengtaElementu: Integer;
    suminisPadengimas: TPadengimoVektorius;
    Kaina: Real;
begin
  Kaina:= 0;
  suminisPadengimas:= TPadengimoVektorius.Create;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    suminisPadengimas.Add;
  for i:= 0 to Chromosoma.Genai.Count - 1 do
    if Chromosoma.Genai[i].Reiksme > 0 then
    begin
      for j:= 0 to Siustuvai.Zonos.Count - 1 do
        suminisPadengimas[j].Padengta:= suminisPadengimas[j].Padengta +
                                        Siustuvai[i].PadengimoVektorius[j].Padengta;
      Kaina:= Kaina + Siustuvai[i].SiustuvoKaina + Siustuvai[i].ZemesKaina;
    end;
  visoElementu:= 0;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    visoElementu:= visoElementu + Siustuvai.Zonos[i].PadengimoLaipsnis;
  padengtaElementu:= 0;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    if suminisPadengimas[i].Padengta < Siustuvai.Zonos[i].PadengimoLaipsnis then
      padengtaElementu:= padengtaElementu + suminisPadengimas[i].Padengta
    else
      padengtaElementu:= padengtaElementu + Siustuvai.Zonos[i].PadengimoLaipsnis;
  if visoElementu = 0 then
    if Kaina = 0 then
      Result:= 1
    else
      Result:= 1 / Kaina
  else
    if Kaina = 0 then
      Result:= padengtaElementu / visoElementu
    else
      Result:= Power(padengtaElementu / visoElementu, 4) / Kaina;
  suminisPadengimas.Free;
  suminisPadengimas:= nil;
end;

procedure TBevielesPrieigosAlgoritmas.IsrikiuotiPopuliacija(naujaPopuliacija: TPopuliacija);
var i, j, numMaxKokybe: Integer;
    kokybe, maxKokybe: Real;
begin
  for i:= 0 to naujaPopuliacija.Count - 1 do
  begin
    maxKokybe:= naujaPopuliacija[i].Kokybe;
    numMaxKokybe:= i;
    for j:= i to naujaPopuliacija.Count - 1 do
      if naujaPopuliacija[j].Kokybe > maxKokybe then
      begin
        maxKokybe:= naujaPopuliacija[j].Kokybe;
        numMaxKokybe:= j;
      end;
    if numMaxKokybe <> i then
    begin
      naujaPopuliacija[numMaxKokybe].Index:= i;
      naujaPopuliacija[i + 1].Index:= numMaxKokybe - 1;
    end;
  end;
end;

procedure TBevielesPrieigosAlgoritmas.ElitoKopijavimas(naujaPopuliacija: TPopuliacija;
                                                       ElitoKoef: Real);
var i: Integer;
begin
  for i:= 0 to Round(Populiacija.Count * ElitoKoef) - 1 do
    naujaPopuliacija.Add.Assign(Populiacija[i]);
end;

procedure TBevielesPrieigosAlgoritmas.Mutavimas(naujaPopuliacija: TPopuliacija;
                                                MutavimoKoef: Real);
var i: Integer;
begin
  while i < Round(naujaPopuliacija.Count * MutavimoKoef) do
  begin
    GeneruotiGenus(naujaPopuliacija[naujaPopuliacija.Count - i - 1]);
    i:= i + 1;
  end;
```

```
end;

function TBevielesPrieigosAlgoritmas.AtrinktiTeva(Tevas: Integer): TChromosoma;
var i: Integer;
    sum, sum2, kriterijus: real;
begin
  sum:= 0;
  for i:= 0 to Populiacija.Count - 1 do
    if i <> Tevas then
      sum:= sum + Populiacija[i].Kokybe;
  Randomize;
  kriterijus:= Random;
  i:= 0;
  sum2:= 0;
  while (sum2 <= kriterijus) and (i < Populiacija.Count) do
  begin
    if i <> Tevas then
      sum2:= sum2 + Populiacija[i].Kokybe / sum;
    i:= i + 1;
  end;
  Result:= Populiacija[i - 1];
end;

procedure TBevielesPrieigosAlgoritmas.DvitaskisKryzminimas(naujaPopuliacija: TPopuliacija);
var i, j: Integer;
    taskas, taskas2: Integer;
    vaikas1, vaikas2, tevas1, tevas2: TChromosoma;
begin
  i:= naujaPopuliacija.Count;
  while i < Populiacija.Count do
  begin
    tevas1:= AtrinktiTeva(-1);
    tevas2:= AtrinktiTeva(tevas1.Index);
    vaikas1:= naujaPopuliacija.Add;
    vaikas2:= naujaPopuliacija.Add;
    Randomize;
    taskas:= Random(tevas1.Genai.Count);
    taskas2:= Random(tevas1.Genai.Count);
    for j:= 0 to tevas1.Genai.Count - 1 do
      if (((j <= taskas) or (j > taskas2)) and (taskas2 > taskas)) or
         (((j <= taskas2) or (j > taskas)) and (taskas >= taskas2)) then
      begin
        vaikas1.Genai.Add.Assign(tevas1.Genai[j]);
        vaikas2.Genai.Add.Assign(tevas2.Genai[j]);
      end
      else
      begin
        vaikas1.Genai.Add.Assign(tevas2.Genai[j]);
        vaikas2.Genai.Add.Assign(tevas1.Genai[j]);
      end;
     if i = Populiacija.Count - 1 then
     begin
       Randomize;
       if Random < 0.5 then
         naujaPopuliacija.Delete(naujaPopuliacija.Count - 2)
       else
         naujaPopuliacija.Delete(naujaPopuliacija.Count - 1);
     end;
    i:= i + 2;
  end;
end;

function TBevielesPrieigosAlgoritmas.AtnaujintiPopuliacija(PopuliacijosDydis: Integer;
                                                           MutavimoKoef: Real;
                                                           ElitoKoef: Real;
                                                           NaudojamaLokaliPaieska: boolean): boolean;
var naujaPopuliacija: TPopuliacija;
    i, mutavimoSk, elitoSk: Integer;
begin
  naujaPopuliacija:= TPopuliacija.Create;
  ElitoKopijavimas(naujaPopuliacija, ElitoKoef);
  DvitaskisKryzminimas(naujaPopuliacija);
  for i:= Round(naujaPopuliacija.Count * ElitoKoef) to naujaPopuliacija.Count - 1 do
    naujaPopuliacija[i].Kokybe:= ChromosomosKokybe(naujaPopuliacija[i]);
  IsrikiuotiPopuliacija(naujaPopuliacija);
  Mutavimas(naujaPopuliacija, MutavimoKoef);
  if NaudojamaLokaliPaieska then
    LokaliPaieska(0, naujaPopuliacija);
  for i:= 0 to naujaPopuliacija.Count - 1 do
    naujaPopuliacija[i].Kokybe:= ChromosomosKokybe(naujaPopuliacija[i]);
```

```
  IsrikiuotiPopuliacija(naujaPopuliacija);
  Populiacija.Assign(naujaPopuliacija);
  naujaPopuliacija.Free;
  naujaPopuliacija:= nil;
  Result:= true;
end;

function TBevielesPrieigosAlgoritmas.GenetinisAlgoritmas(IteracijuSk, PopuliacijosDydis: Integer;
                                                         NaudojamaLokaliPaieska: boolean;
                                                         MutavimoKoef: Real;
                                                         ElitoKoef: Real;
                                                         out     maxKokybesChromosoma:     TChromosoma):
boolean;
var i: Integer;
begin
  maxKokybesChromosoma:= TChromosoma.Create(nil);
  GeneruotiPradinePopuliacija(PopuliacijosDydis);
  if NaudojamaLokaliPaieska then
    LokaliPaieska(0, Populiacija);
  for i:= 0 to Populiacija.Count - 1 do
    Populiacija[i].Kokybe:= ChromosomosKokybe(Populiacija[i]);
  IsrikiuotiPopuliacija(Populiacija);
  maxKokybesChromosoma.Assign(Populiacija[0]);
  for i:= 1 to IteracijuSk do
  begin
    if not AtnaujintiPopuliacija(PopuliacijosDydis, MutavimoKoef, ElitoKoef,
                                 NaudojamaLokaliPaieska) then
    begin
      Result:= false;
      Exit;
    end;
    if maxKokybesChromosoma.Kokybe < Populiacija[0].Kokybe then
      maxKokybesChromosoma.Assign(Populiacija[0]);
  end;
  Result:= true;
end;

function TBevielesPrieigosAlgoritmas.TinkloKastai(Sprendinys: TSiustuvaiList): real;
var i: integer;
begin
  Result:= 0;
  for i:= 0 to Sprendinys.Count - 1 do
    Result:= Result + Sprendinys[i].SiustuvoKaina + Sprendinys[i].ZemesKaina;
end;

function TBevielesPrieigosAlgoritmas.Apskaiciuoti(Siustuvai1: TSiustuvaiList;
                                                  Parametrai: TMetodikosParamList;
                                                  Log: TStrings;
                                                  out Sprendinys: TSiustuvaiList;
                                                  out Kaina, Padengta: Real): boolean;
var i, j: integer;
    FormatSet: TFormatSettings;
    sprendinioChromosoma: TChromosoma;
begin
  Logas:= Log;
  Siustuvai:= Siustuvai1;
  GetLocaleFormatSettings(1033, FormatSet);
  MinLeistinasPadengimas:= StrToFloat(Parametrai[5].Reiksme, FormatSet);
  if GenetinisAlgoritmas(StrToInt(Parametrai[0].Reiksme),
                         StrToInt(Parametrai[1].Reiksme),
                         boolean(StrToInt(Parametrai[3].Reiksme)),
                         StrToFloat(Parametrai[2].Reiksme, FormatSet),
                         StrToFloat(Parametrai[4].Reiksme, FormatSet),
                         sprendinioChromosoma) then
  begin
    Sprendinys:= TSiustuvaiList.Create;
    Sprendinys.KopijuotiSiustuvuParametrus(Siustuvai);
    for i:= 0 to sprendinioChromosoma.Genai.Count - 1 do
      if sprendinioChromosoma.Genai[i].Reiksme > 0 then
      begin
        Sprendinys.Add.Assign(Siustuvai[i]);
        Sprendinys[Sprendinys.Count - 1].AptarnaujamosZonos.KopijuotiZonuParametrus(Siustuvai.Zonos);
        for j:= 0 to Siustuvai[i].PadengimoVektorius.Count - 1 do
          if Siustuvai[i].PadengimoVektorius[j].Padengta > 0 then
            Sprendinys[Sprendinys.Count - 1].AptarnaujamosZonos.Add.Assign(Siustuvai.Zonos[j]);
      end;
    Kaina:= TinkloKastai(Sprendinys);
    Padengta:= PadengimoProcentas(sprendinioChromosoma) * 100;
    if Padengta / 100 < StrToFloat(Parametrai[5].Reiksme, FormatSet) then
      Result:= false
```

```
    else
      Result:= true;
  end
  else
    Result:= false;
  Siustuvai:= nil;
  Logas:= nil;
  Populiacija.Free;
  Populiacija:= nil;
  sprendinioChromosoma.Free;
  sprendinioChromosoma:= nil;
end;


end.
```

unit LaidinesPrieigosAlgoritmas; //Genetinis algoritmas

```
interface

uses
  Classes, SysUtils, Zonos, Siustuvai, Math, StociuMetodikosInfo, PadengimoVektorius,
  StdCtrls, DuomenuStrukturos;

type
  TLaidinesPrieigosAlgoritmas = class
  private
    Populiacija: TPopuliacijaL;
    Siustuvai: TSiustuvaiList;
    Logas: TStrings;
    procedure GeneruotiChromosoma(Individas: TIndividasL; Chromosoma: TChromosomaL);
    procedure GeneruotiChromosomas(Individas: TIndividasL);
    procedure GeneruotiPradinePopuliacija(Dydis: Integer);
    procedure IsrikiuotiPopuliacija(naujaPopuliacija: TPopuliacijaL);
    procedure DvitaskisKryzminimas(naujaPopuliacija: TPopuliacijaL);
    procedure LokaliPaieska(Nuo: Integer; naujaPopuliacija: TPopuliacijaL);
    procedure LokaliPaieskaIndividui(Individas: TIndividasL);
    procedure ElitoKopijavimas(naujaPopuliacija: TPopuliacijaL; ElitoKoef: Real);
    procedure Mutavimas(naujaPopuliacija: TPopuliacijaL; MutavimoKoef: Real);
    procedure PataisytiIndivida(Individas: TIndividasL);
    procedure AtnaujintiReiksmes(Individas: TIndividasL);
    procedure PerskirstytiZonas(Individas: TIndividasL; num: Integer);
    function PadengimoProcentas(Individas: TIndividasL): Real;
    function ArGalimaPasalinti(num: Integer; Individas: TIndividasL): boolean;
    function AtrinktiTeva(Tevas: Integer): TIndividasL;
    function AtnaujintiPopuliacija(PopuliacijosDydis: Integer;
                                   MutavimoKoef: Real;
                                   ElitoKoef: Real;
                                   NaudojamaLokaliPaieska: boolean): boolean;
    function IndividoKokybe(Individas: TIndividasL): Real;
    function GenetinisAlgoritmas(IteracijuSk, PopuliacijosDydis: Integer;
                                 NaudojamaLokaliPaieska: boolean;
                                 MutavimoKoef: Real; ElitoKoef: Real;
                                 out maxKokybesIndividas: TIndividasL): boolean;
    function TinkloKastai(Sprendinys: TSiustuvaiList): real;
    function SiustuvoKaina(Siustuvas: TSiustuvaiListItem): Real;
    function PrijungimoKastai(ZoneIndex: Integer;
                              Siustuvas: TSiustuvaiListItem): Real;
    function ChromosomosKaina(Chromosoma: TChromosomaL): Real;
    function IsmestiChromosoma(Num: Integer; Individas: TIndividasL): boolean;
  protected

  public
    MinLeistinasPadengimas: Real;
    function Apskaiciuoti(Siustuvail: TSiustuvaiList; Parametrai: TMetodikosParamList;
                          Log: TStrings;
                          out Sprendinys: TSiustuvaiList; out Kaina, Padengta: real): boolean;
  end;

implementation

function TLaidinesPrieigosAlgoritmas.IsmestiChromosoma(Num: Integer;
                                                       Individas: TIndividasL): boolean;
var i, j: Integer;
    priemimoKastai, minPriemimoKastai, minPriemimoKastaiSum: Real;
    priimancioNum: array of Integer;
begin
  SetLength(priimancioNum, Siustuvai.Zonos.Count);
  minPriemimoKastaiSum:= 0;
```

```
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    if Individas.Chromosomos[Num].Genai[i].Padengta > 0 then
    begin
      minPriemimoKastai:= MaxExtended;
      priimancioNum[i]:= -1;
      for j:= 0 to Siustuvai.Count - 1 do
        if Individas.Chromosomos[j].Reiksme > 0 then
          if (j <> Num) and
             (Siustuvai[j].PadengimoVektorius[i].Padengta > 0) then
          begin
            priemimoKastai:= PrijungimoKastai(i, Siustuvai[j]);
            if priemimoKastai < minPriemimoKastai then
            begin
              minPriemimoKastai:= priemimoKastai;
              priimancioNum[i]:= j;
            end;
          end;
      minPriemimoKastaiSum:= minPriemimoKastaiSum + minPriemimoKastai;
    end;
  if ChromosomosKaina(Individas.Chromosomos[Num]) > minPriemimoKastaiSum then
  begin
    for i:= 0 to Siustuvai.Zonos.Count - 1 do
      if Individas.Chromosomos[Num].Genai[i].Padengta > 0 then
      begin
        Individas.Chromosomos[priimancioNum[i]].Genai[i].Padengta:= 1;
        Individas.Chromosomos[Num].Genai[i].Padengta:= 0;
      end;
    Result:= true;
  end
  else
    Result:= false;
  PriimancioNum:= nil;
end;

function TLaidinesPrieigosAlgoritmas.ArGalimaPasalinti(num: Integer;
                                                       Individas: TIndividasL): boolean;
var suminisPadengimas: TPadengimoVektorius;
    i, j: Integer;
begin
  suminisPadengimas:= TPadengimoVektorius.Create;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    suminisPadengimas.Add;
  for i:= 0 to Siustuvai.Count - 1 do
    if i <> num then
      for j:= 0 to Siustuvai.Zonos.Count - 1 do
          suminisPadengimas[j].Padengta:= suminisPadengimas[j].Padengta +
                                          Siustuvai[i].PadengimoVektorius[j].Padengta;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    if suminisPadengimas[i].Padengta < Siustuvai.Zonos[i].PadengimoLaipsnis then
    begin
      Result:= false;
      suminisPadengimas.Free;
      suminisPadengimas:= nil;
      Exit;
    end;
  Result:= true;
  suminisPadengimas.Free;
  suminisPadengimas:= nil;
end;

procedure TLaidinesPrieigosAlgoritmas.PerskirstytiZonas(Individas: TIndividasL; Num: Integer);
var i, j: Integer;
    priemimoKastai, minPriemimoKastai: Real;
    nemetam: boolean;
    priimancioNum: array of Integer;
begin
  SetLength(priimancioNum, Siustuvai.Zonos.Count);
  nemetam:= false;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    if Individas.Chromosomos[Num].Genai[i].Padengta > 0 then
    begin
      minPriemimoKastai:= PrijungimoKastai(i, Siustuvai[Num]);
      priimancioNum[i]:= -1;
      for j:= 0 to Siustuvai.Count - 1 do
        if Individas.Chromosomos[j].Reiksme > 0 then
          if (j <> Num) and
             (Siustuvai[j].PadengimoVektorius[i].Padengta > 0) then
          begin
            priemimoKastai:= PrijungimoKastai(i, Siustuvai[j]);
            if priemimoKastai < minPriemimoKastai then
```

```
            begin
              minPriemimoKastai:= priemimoKastai;
              priimancioNum[i]:= j;
            end;
          end;
      if priimancioNum[i] > -1 then
      begin
        Individas.Chromosomos[priimancioNum[i]].Genai[i].Padengta:= 1;
        Individas.Chromosomos[Num].Genai[i].Padengta:= 0;
      end
      else
        nemetam:= true;
    end;
  if not nemetam then
    Individas.Chromosomos[Num].Reiksme:= 0;
  PriimancioNum:= nil;
end;

procedure TLaidinesPrieigosAlgoritmas.LokaliPaieskaIndividui(Individas: TIndividasL);
var i: Integer;
begin
  for i:= 0 to Individas.Chromosomos.Count - 1 do
    PerskirstytiZonas(Individas, i);
  for i:= 0 to Individas.Chromosomos.Count - 1 do
    if Individas.Chromosomos[i].Reiksme > 0 then
      if ArGalimaPasalinti(i, Individas) then
        if IsmestiChromosoma(i, Individas) then
          Individas.Chromosomos[i].Reiksme:= 0;
end;

procedure TLaidinesPrieigosAlgoritmas.LokaliPaieska(Nuo: Integer; naujaPopuliacija: TPopuliacijaL);
var i: Integer;
begin
  for i:= Nuo to naujaPopuliacija.Count - 1 do
    LokaliPaieskaIndividui(naujaPopuliacija[i]);
end;

procedure    TLaidinesPrieigosAlgoritmas.GeneruotiChromosoma(Individas:    TIndividasL;    Chromosoma:
TChromosomaL);
var i: Integer;
begin
  while Chromosoma.Genai.Count < Siustuvai.Zonos.Count do
    Chromosoma.Genai.Add;
  Randomize;
  if Random < 0.5 then
    Exit;
  for i:= 0 to Chromosoma.Genai.Count - 1 do
    if Siustuvai[Chromosoma.Index].PadengimoVektorius[i].Padengta > 0 then
      if Chromosoma.Genai[i].Padengta < Siustuvai.Zonos[i].PadengimoLaipsnis then
      begin
        Randomize;
        if Random >= 0.5 then
        begin
          Chromosoma.Genai[i].Padengta:= 1;
          Chromosoma.Reiksme:= 1;
        end;
      end;
end;

procedure TLaidinesPrieigosAlgoritmas.GeneruotiChromosomas(Individas: TIndividasL);
var i: Integer;
    oldProc, proc: real;
begin
  Individas.Chromosomos.Clear;
  while Individas.Chromosomos.Count < Siustuvai.Count do
    Individas.Chromosomos.Add;
  repeat
    oldProc := PadengimoProcentas(Individas);
    for i := 0 to Siustuvai.Count - 1 do
      GeneruotiChromosoma(Individas, Individas.Chromosomos[i]);
    proc := PadengimoProcentas(Individas);
  until (proc >= MinLeistinasPadengimas) or (oldProc >= proc);
end;

procedure TLaidinesPrieigosAlgoritmas.GeneruotiPradinePopuliacija(Dydis: Integer);
var Individas: TIndividasL;
begin
  if Populiacija = nil then
    Populiacija:= TPopuliacijaL.Create
  else
```

```
    Populiacija.Clear;
  while Populiacija.Count < Dydis do
  begin
    Individas:= Populiacija.Add;
    GeneruotiChromosomas(Individas);
    PataisytiIndivida(Individas);
    AtnaujintiReiksmes(Individas);
    Individas:= nil;
  end;
end;

function TLaidinesPrieigosAlgoritmas.PadengimoProcentas(Individas: TIndividasL): Real;
var i, j, padengtaElementu, visoElementu: Integer;
    suminisPadengimas: TPadengimoVektorius;
begin
  suminisPadengimas:= TPadengimoVektorius.Create;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    suminisPadengimas.Add;
  for i:= 0 to Siustuvai.Count - 1 do
    if Individas.Chromosomos[i].Reiksme > 0 then
      for j:= 0 to Siustuvai.Zonos.Count - 1 do
        suminisPadengimas[j].Padengta:= suminisPadengimas[j].Padengta +
                                        Individas.Chromosomos[i].Genai[j].Padengta;
  padengtaElementu:= 0;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    if suminisPadengimas[i].Padengta < Siustuvai.Zonos[i].PadengimoLaipsnis then
      padengtaElementu:= padengtaElementu + suminisPadengimas[i].Padengta
    else
      padengtaElementu:= padengtaElementu + Siustuvai.Zonos[i].PadengimoLaipsnis;
  visoElementu:= 0;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    visoElementu:= visoElementu + Siustuvai.Zonos[i].PadengimoLaipsnis;
  if visoElementu = 0 then
    Result:= 1
  else
    Result:= padengtaElementu / visoElementu;
  suminisPadengimas.Free;
  suminisPadengimas:= nil;
end;

function TLaidinesPrieigosAlgoritmas.IndividoKokybe(Individas: TIndividasL): Real;
var i, j, padengtaElementu, visoElementu: Integer;
    suminisPadengimas: TPadengimoVektorius;
    Kaina: Real;
begin
  Kaina:= 0;
  suminisPadengimas:= TPadengimoVektorius.Create;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    suminisPadengimas.Add;
  for i:= 0 to Siustuvai.Count - 1 do
    if Individas.Chromosomos[i].Reiksme > 0 then
    begin
      for j:= 0 to Siustuvai.Zonos.Count - 1 do
        suminisPadengimas[j].Padengta:= suminisPadengimas[j].Padengta +
                                        Individas.Chromosomos[i].Genai[j].Padengta;
      Kaina:= Kaina + ChromosomosKaina(Individas.Chromosomos[i]);
    end;
  padengtaElementu:= 0;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    if suminisPadengimas[i].Padengta < Siustuvai.Zonos[i].PadengimoLaipsnis then
      padengtaElementu:= padengtaElementu + suminisPadengimas[i].Padengta
    else
      padengtaElementu:= padengtaElementu + Siustuvai.Zonos[i].PadengimoLaipsnis;
  visoElementu:= 0;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    visoElementu:= visoElementu + Siustuvai.Zonos[i].PadengimoLaipsnis;
  if visoElementu = 0 then
    if Kaina = 0 then
      Result:= 1
    else
      Result:= 1 / Kaina
  else
    if Kaina = 0 then
      Result:= padengtaElementu / visoElementu
    else
      Result:= Power(padengtaElementu / visoElementu, 4) / Kaina;
  suminisPadengimas.Free;
  suminisPadengimas:= nil;
end;
```

```
procedure TLaidinesPrieigosAlgoritmas.IsrikiuotiPopuliacija(naujaPopuliacija: TPopuliacijaL);
var i, j, numMaxKokybe: Integer;
    kokybe, maxKokybe: Real;
begin
  for i:= 0 to naujaPopuliacija.Count - 1 do
  begin
    maxKokybe:= naujaPopuliacija[i].Kokybe;
    numMaxKokybe:= i;
    for j:= i to naujaPopuliacija.Count - 1 do
      if naujaPopuliacija[j].Kokybe > maxKokybe then
      begin
        maxKokybe:= naujaPopuliacija[j].Kokybe;
        numMaxKokybe:= j;
      end;
    if numMaxKokybe <> i then
    begin
      naujaPopuliacija[numMaxKokybe].Index:= i;
      naujaPopuliacija[i + 1].Index:= numMaxKokybe - 1;
    end;
  end;
end;

procedure TLaidinesPrieigosAlgoritmas.ElitoKopijavimas(naujaPopuliacija: TPopuliacijaL;
                                                       ElitoKoef: Real);
var i: Integer;
begin
  for i:= 0 to Round(Populiacija.Count * ElitoKoef) - 1 do
    naujaPopuliacija.Add.Assign(Populiacija[i]);
end;

procedure TLaidinesPrieigosAlgoritmas.AtnaujintiReiksmes(Individas: TIndividasL);
var i, j, sum: Integer;
begin
  for i:= 0 to Siustuvai.Count - 1 do
  begin
    sum:= 0;
    for j:= 0 to Siustuvai.Zonos.Count - 1 do
      sum:= sum + Individas.Chromosomos[i].Genai[j].Padengta;
    if sum = 0 then
      Individas.Chromosomos[i].Reiksme:= 0;
  end;
end;

procedure TLaidinesPrieigosAlgoritmas.Mutavimas(naujaPopuliacija: TPopuliacijaL;
                                                MutavimoKoef: Real);
var i: Integer;
begin
  while i < Round(naujaPopuliacija.Count * MutavimoKoef) do
  begin
    GeneruotiChromosomas(naujaPopuliacija[naujaPopuliacija.Count - i - 1]);
    PataisytiIndivida(naujaPopuliacija[naujaPopuliacija.Count - i - 1]);
    AtnaujintiReiksmes(naujaPopuliacija[naujaPopuliacija.Count - i - 1]);
    i:= i + 1;
  end;
end;

function TLaidinesPrieigosAlgoritmas.AtrinktiTeva(Tevas: Integer): TIndividasL;
var i: Integer;
    sum, sum2, kriterijus: real;
begin
  sum:= 0;
  for i:= 0 to Populiacija.Count - 1 do
    if i <> Tevas then
      sum:= sum + Populiacija[i].Kokybe;
  Randomize;
  kriterijus:= Random;
  i:= 0;
  sum2:= 0;
  while (sum2 <= kriterijus) and (i < Populiacija.Count) do
  begin
    if i <> Tevas then
      sum2:= sum2 + Populiacija[i].Kokybe / sum;
    i:= i + 1;
  end;
  Result:= Populiacija[i - 1];
end;

procedure TLaidinesPrieigosAlgoritmas.PataisytiIndivida(Individas: TIndividasL);
var i, j, num: Integer;
    Kastai, minKastai: Real;
```

```
begin
  for i:= 0 to Siustuvai.Zonos.Count -  1 do
  begin
    minKastai:= MaxExtended;
    num:= -1;
    for j:= 0 to Siustuvai.Count - 1 do
      if Individas.Chromosomos[j].Reiksme > 0 then
        if Individas.Chromosomos[j].Genai[i].Padengta > 0 then
        begin
          Kastai:= PrijungimoKastai(i, Siustuvai[j]);
          if Kastai < minKastai then
          begin
            minKastai:= Kastai;
            num:= j;
          end;
        end;
    if num > -1 then
      for j:= 0 to Siustuvai.Count - 1 do
        if j <> num then
          Individas.Chromosomos[j].Genai[i].Padengta:= 0;
  end;
end;

procedure TLaidinesPrieigosAlgoritmas.DvitaskisKryzminimas(naujaPopuliacija: TPopuliacijaL);
var i, j: Integer;
    taskas, taskas2: Integer;
    vaikas1, vaikas2, tevas1, tevas2: TIndividasL;
begin
  i:= naujaPopuliacija.Count;
  while i < Populiacija.Count do
  begin
    tevas1:= AtrinktiTeva(-1);
    tevas2:= AtrinktiTeva(tevas1.Index);
    vaikas1:= naujaPopuliacija.Add;
    vaikas2:= naujaPopuliacija.Add;
    Randomize;
    taskas:= Random(tevas1.Chromosomos.Count);
    taskas2:= Random(tevas1.Chromosomos.Count);
    for j:= 0 to tevas1.Chromosomos.Count - 1 do
       if (((j <= taskas) or (j > taskas2)) and (taskas2 > taskas)) or
          (((j <= taskas2) or (j > taskas)) and (taskas >= taskas2)) then
      begin
        vaikas1.Chromosomos.Add.Assign(tevas1.Chromosomos[j]);
        vaikas2.Chromosomos.Add.Assign(tevas2.Chromosomos[j]);
      end
      else
      begin
        vaikas1.Chromosomos.Add.Assign(tevas2.Chromosomos[j]);
        vaikas2.Chromosomos.Add.Assign(tevas1.Chromosomos[j]);
      end;
    if i = Populiacija.Count - 1 then
    begin
      Randomize;
      if Random < 0.5 then
      begin
        PataisytiIndivida(vaikas2);
        naujaPopuliacija.Delete(naujaPopuliacija.Count - 2);
      end
      else
      begin
        PataisytiIndivida(vaikas1);
        naujaPopuliacija.Delete(naujaPopuliacija.Count - 1);
      end;
    end
    else
    begin
      PataisytiIndivida(vaikas1);
      PataisytiIndivida(vaikas2);
    end;
    i:= i + 2;
  end;
end;

function TLaidinesPrieigosAlgoritmas.AtnaujintiPopuliacija(PopuliacijosDydis: Integer;
                                                           MutavimoKoef: Real;
                                                           ElitoKoef: Real;
                                                           NaudojamaLokaliPaieska: boolean): boolean;
var naujaPopuliacija: TPopuliacijaL;
    i, mutavimoSk, elitoSk: Integer;
begin
```

```
  naujaPopuliacija:= TPopuliacijaL.Create;
  ElitoKopijavimas(naujaPopuliacija, ElitoKoef);
  DvitaskisKryzminimas(naujaPopuliacija);
  for i:= Round(naujaPopuliacija.Count * ElitoKoef) to naujaPopuliacija.Count - 1 do
    Populiacija[i].Kokybe:= IndividoKokybe(Populiacija[i]);
  IsrikiuotiPopuliacija(naujaPopuliacija);
  Mutavimas(naujaPopuliacija, MutavimoKoef);
  if NaudojamaLokaliPaieska then
    LokaliPaieska(0, naujaPopuliacija);
  for i:= 0 to naujaPopuliacija.Count - 1 do
    naujaPopuliacija[i].Kokybe:= IndividoKokybe(naujaPopuliacija[i]);
  IsrikiuotiPopuliacija(naujaPopuliacija);
  Populiacija.Assign(naujaPopuliacija);
  naujaPopuliacija.Free;
  naujaPopuliacija:= nil;
  Result:= true;
end;

function TLaidinesPrieigosAlgoritmas.GenetinisAlgoritmas(IteracijuSk, PopuliacijosDydis: Integer;
                                                         NaudojamaLokaliPaieska: boolean;
                                                         MutavimoKoef: Real;
                                                         ElitoKoef: Real;
                                                         out     maxKokybesIndividas:     TIndividasL):
boolean;
var i: Integer;
begin
  maxKokybesIndividas:= TIndividasL.Create(nil);
  GeneruotiPradinePopuliacija(PopuliacijosDydis);
  if NaudojamaLokaliPaieska then
    LokaliPaieska(0, Populiacija);
  for i:= 0 to Populiacija.Count - 1 do
    Populiacija[i].Kokybe:= IndividoKokybe(Populiacija[i]);
  IsrikiuotiPopuliacija(Populiacija);
  maxKokybesIndividas.Assign(Populiacija[0]);
  for i:= 1 to IteracijuSk do
  begin
    if not AtnaujintiPopuliacija(PopuliacijosDydis, MutavimoKoef, ElitoKoef,
                                 NaudojamaLokaliPaieska) then
    begin
      Result:= false;
      Exit;
    end;
    if maxKokybesIndividas.Kokybe < Populiacija[0].Kokybe then
      maxKokybesIndividas.Assign(Populiacija[0]);
  end;
  Result:= true;
end;

function TLaidinesPrieigosAlgoritmas.PrijungimoKastai(ZoneIndex: Integer;
                                                      Siustuvas: TSiustuvaiListItem): Real;
begin
  Result:= Siustuvai.Zonos[ZoneIndex].VartotojuSkaicius *
            Siustuvas.PrijungimoKastaiKm * sqrt(sqr(Siustuvas.X -
            Siustuvai.Zonos[ZoneIndex].X - Siustuvai.Zonos[ZoneIndex].Ilgis / 2) +
            sqr(Siustuvas.Y - Siustuvai.Zonos[ZoneIndex].Y -
            Siustuvai.Zonos[ZoneIndex].Ilgis / 2))
end;

function TLaidinesPrieigosAlgoritmas.ChromosomosKaina(Chromosoma: TChromosomaL): Real;
var i: Integer;
begin
  Result:= Siustuvai[Chromosoma.Index].SiustuvoKaina + Siustuvai[Chromosoma.Index].ZemesKaina;
  for i:= 0 to Chromosoma.Genai.Count - 1 do
    if Chromosoma.Genai[i].Padengta > 0 then
      Result:= Result + PrijungimoKastai(i, Siustuvai[Chromosoma.Index]);
end;

function TLaidinesPrieigosAlgoritmas.SiustuvoKaina(Siustuvas: TSiustuvaiListItem): Real;
var i: Integer;
begin
  Result:= Siustuvas.SiustuvoKaina + Siustuvas.ZemesKaina;
  for i:= 0 to Siustuvas.AptarnaujamosZonos.Count - 1 do
    Result:= Result + Siustuvas.AptarnaujamosZonos[i].PrijungimoKastai;
end;

function TLaidinesPrieigosAlgoritmas.TinkloKastai(Sprendinys: TSiustuvaiList): real;
var i: integer;
begin
  Result:= 0;
  for i:= 0 to Sprendinys.Count - 1 do
```

```
    Result:= Result + SiustuvoKaina(Sprendinys[i]);
end;

function TLaidinesPrieigosAlgoritmas.Apskaiciuoti(Siustuvai1: TSiustuvaiList;
                                                  Parametrai: TMetodikosParamList;
                                                  Log: TStrings;
                                                  out Sprendinys: TSiustuvaiList;
                                                  out Kaina, Padengta: Real): boolean;
var i, j: integer;
    FormatSet: TFormatSettings;
    sprendinioIndividas: TIndividasL;
begin
  Logas:= Log;
  Siustuvai:= Siustuvai1;
  GetLocaleFormatSettings(1033, FormatSet);
  MinLeistinasPadengimas:= StrToFloat(Parametrai[5].Reiksme, FormatSet);
  if GenetinisAlgoritmas(StrToInt(Parametrai[0].Reiksme),
                         StrToInt(Parametrai[1].Reiksme),
                         boolean(StrToInt(Parametrai[3].Reiksme)),
                         StrToFloat(Parametrai[2].Reiksme, FormatSet),
                         StrToFloat(Parametrai[4].Reiksme, FormatSet),
                         sprendinioIndividas) then
  begin
    Sprendinys:= TSiustuvaiList.Create;
    Sprendinys.KopijuotiSiustuvuParametrus(Siustuvai);
    for i:= 0 to sprendinioIndividas.Chromosomos.Count - 1 do
      if sprendinioIndividas.Chromosomos[i].Reiksme > 0 then
      begin
        Sprendinys.Add.Assign(Siustuvai[i]);
        Sprendinys[Sprendinys.Count - 1].AptarnaujamosZonos.KopijuotiZonuParametrus(Siustuvai.Zonos);
        for j:= 0 to sprendinioIndividas.Chromosomos[i].Genai.Count - 1 do
          if sprendinioIndividas.Chromosomos[i].Genai[j].Padengta > 0 then
          begin
            Sprendinys[Sprendinys.Count - 1].AptarnaujamosZonos.Add.Assign(Siustuvai.Zonos[j]);
            Sprendinys[Sprendinys.Count     -     1].AptarnaujamosZonos[Sprendinys[Sprendinys.Count     -
1].AptarnaujamosZonos.Count - 1].PrijungimoKastai:=
            PrijungimoKastai(Sprendinys[Sprendinys.Count                                                -
1].AptarnaujamosZonos[Sprendinys[Sprendinys.Count - 1].AptarnaujamosZonos.Count - 1].Identifikatorius,
                             Sprendinys[Sprendinys.Count - 1]);
          end;
      end;
    Kaina:= TinkloKastai(Sprendinys);
    Padengta:= PadengimoProcentas(sprendinioIndividas) * 100;
    if Padengta / 100 < StrToFloat(Parametrai[5].Reiksme, FormatSet) then
      Result:= false
    else
      Result:= true;
  end
  else
    Result:= false;
  Siustuvai:= nil;
  Logas:= nil;
  Populiacija.Free;
  Populiacija:= nil;
  sprendinioIndividas.Free;
  sprendinioIndividas:= nil;
end;


end.
```

unit BevielesPrieigosAlgoritmas;

```
interface

uses
  Zonos, Siustuvai, Math, StociuMetodikosInfo, PadengimoVektorius, Classes,
  DuomenuStrukturos, SysUtils;

type

  TBevielesPrieigosAlgoritmas = class
  private
    Logas: TStringList;
    Siustuvai: TSiustuvaiList;
    procedure LokaliPaieska(Solution: TSiustuvaiList);
    procedure AtnaujintiFeromonus(PartialSolutions: TPartialSolutionsList;
                                  Ro: real);
    procedure AtnaujintiGeriausiaSprendini(PartialSolutions: TPartialSolutionsList;
```

```
                                        out Solution: TSiustuvaiList);
    function PadengiamosZonos(Siustuvas: TSiustuvaiListItem): Integer;
    function PapildytiSprendini(PartialSolution: TSiustuvaiList;
                                Beta: Real): boolean;
    function SkruzdziuKolonijosAlgoritmas(IterCount, AntsCount: integer;
                                          NaudojamaLokaliPaieska: boolean;
                                          Beta, Ro: Real;
                                          out Solution: TSiustuvaiList): boolean;
    function ArGalimaPasalinti(num: Integer; Solution: TSiustuvaiList): boolean;
    function ArPriklausoSprendiniui(Solution: TSiustuvaiList;
                                    Siustuvas: TSiustuvaiListItem): integer;
    function DaznisSprendiniuose(PartialSolutions: TPartialSolutionsList;
                                 Siustuvas: TSiustuvaiListItem): Real;
    function PatekimoTikimybe(Siustuvas: TSiustuvaiListItem;
                              PartialSolution: TSiustuvaiList; Beta: real): real;
    function ArGautasLeistinasSprendinys(Solution: TSiustuvaiList): boolean;
    function ZonosPadengimoLaipsnis(PartialSolution: TSiustuvaiList;
                                    ZoneIndex: Integer): Integer;
    function PadengiaNaujaZona(PartialSolution: TSiustuvaiList;
                               Siustuvas: TSiustuvaiListItem): boolean;
    function TinkloKastai(Solution: TSiustuvaiList): real;
  protected

  public
    function Apskaiciuoti(PotencialusSiustuvai: TSiustuvaiList;
                          Parametrai: TMetodikosParamList;
                          Log: TStringList;
                          out Sprendinys: TSiustuvaiList;
                          out Kaina, Padengta: real): boolean;
  end;

implementation

function TBevielesPrieigosAlgoritmas.ArGalimaPasalinti(num: Integer;
                                                       Solution: TSiustuvaiList): boolean;
var suminisPadengimas: TPadengimoVektorius;
    i, j: Integer;
begin
  suminisPadengimas:= TPadengimoVektorius.Create;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    suminisPadengimas.Add;
  for i:= 0 to Solution.Count - 1 do
    if i <> num then
      for j:= 0 to Solution[i].PadengimoVektorius.Count - 1 do
        suminisPadengimas[j].Padengta:= suminisPadengimas[j].Padengta +
                                        Solution[i].PadengimoVektorius[j].Padengta;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    if suminisPadengimas[i].Padengta < Siustuvai.Zonos[i].PadengimoLaipsnis then
    begin
      Result:= false;
      suminisPadengimas.Free;
      suminisPadengimas:= nil;
      Exit;
    end;
  Result:= true;
  suminisPadengimas.Free;
  suminisPadengimas:= nil;
end;

procedure TBevielesPrieigosAlgoritmas.LokaliPaieska(Solution: TSiustuvaiList);
var i: Integer;
begin
  i:= 0;
  while i < Solution.Count do
    if ArGalimaPasalinti(i, Solution) then
      Solution.Delete(i)
    else
      i:= i + 1;
end;

function TBevielesPrieigosAlgoritmas.Apskaiciuoti(PotencialusSiustuvai: TSiustuvaiList;
                                                  Parametrai: TMetodikosParamList;
                                                  Log: TStringList;
                                                  out Sprendinys: TSiustuvaiList;
                                                  out Kaina, Padengta: Real): boolean;
var i: integer;
    FormatSet: TFormatSettings;
begin
  Logas:= Log;
  Siustuvai:= PotencialusSiustuvai;
```

```
  for i:= 0 to Siustuvai.Count - 1 do
    Siustuvai[i].Liekanos:= 1 / Siustuvai.Count;
  GetLocaleFormatSettings(1033, FormatSet);
  if SkruzdziuKolonijosAlgoritmas(StrToInt(Parametrai[0].Reiksme),
                                  StrToInt(Parametrai[1].Reiksme),
                                  boolean(StrToInt(Parametrai[4].Reiksme)),
                                  StrToFloat(Parametrai[2].Reiksme, FormatSet),
                                  StrToFloat(Parametrai[3].Reiksme, FormatSet),
                                  Sprendinys) then
  begin
    Sprendinys.KopijuotiSiustuvuParametrus(Siustuvai);
    Kaina:= TinkloKastai(Sprendinys);
    Padengta:= 100;
    Result:= true;
  end
  else
    Result:= false;
  Logas:= nil;
  Siustuvai:= nil;
end;

function TBevielesPrieigosAlgoritmas.ArPriklausoSprendiniui(Solution: TSiustuvaiList;
                                                            Siustuvas: TSiustuvaiListItem): Integer;
var i: integer;
begin
  result:= 0;
  for i:= 0 to Solution.Count - 1 do
    if Solution[i].Identifikatorius = Siustuvas.Identifikatorius then
    begin
      result:= 1;
      exit;
    end;
end;

function TBevielesPrieigosAlgoritmas.ArGautasLeistinasSprendinys(Solution: TSiustuvaiList): boolean;
var i, j, Sum: integer;
begin
 Result:= true;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
  begin
    Sum:= 0;
    for j:= 0 to Solution.Count - 1 do
      Sum:= Sum + Solution[j].PadengimoVektorius[i].Padengta;
    if Sum < Siustuvai.Zonos[i].PadengimoLaipsnis Then
    begin
      Result:= false;
      Exit;
    end;
  end;
end;

function TBevielesPrieigosAlgoritmas.ZonosPadengimoLaipsnis(PartialSolution: TSiustuvaiList;
                                                            ZoneIndex: Integer): Integer;
var i: Integer;
begin
  Result:= 0;
  for i:= 0 to PartialSolution.Count - 1 do
    Result:= Result + PartialSolution[i].PadengimoVektorius[ZoneIndex].Padengta;
end;

function TBevielesPrieigosAlgoritmas.PadengiaNaujaZona(PartialSolution: TSiustuvaiList;
                                       Siustuvas: TSiustuvaiListItem): boolean;
var i: integer;
begin
  Result:= false;
  for i:= 0 to Siustuvas.PadengimoVektorius.Count - 1 do
    if Siustuvas.PadengimoVektorius[i].Padengta > 0 then
      if ZonosPadengimoLaipsnis(PartialSolution, i) <
         Siustuvai.Zonos[i].PadengimoLaipsnis then
        begin
          Result:= true;
          exit;
        end;
end;

function TBevielesPrieigosAlgoritmas.PadengiamosZonos(Siustuvas: TSiustuvaiListItem): Integer;
var i: Integer;
begin
  Result:= 0;
  for i:= 0 to Siustuvas.PadengimoVektorius.Count - 1 do
```

```
    if Siustuvai.Zonos[i].PadengimoLaipsnis > 0 then
      Result:= Result + Siustuvas.PadengimoVektorius[i].Padengta;
end;

function TBevielesPrieigosAlgoritmas.PatekimoTikimybe(Siustuvas: TSiustuvaiListItem;
                                                      PartialSolution: TSiustuvaiList;
                                                      Beta: real): real;
var Vardiklis, Skaitiklis, Dedamoji: real;
    i: integer;
    BelongToFeasibleSolution: boolean;
begin
  Vardiklis:= 0;
  Skaitiklis:= 0;
  BelongToFeasibleSolution:= false;
  for i:= 0 to Siustuvai.Count - 1 do
    if ArPriklausoSprendiniui(PartialSolution, Siustuvai[i]) = 0 then
      if PadengiaNaujaZona(PartialSolution, Siustuvai[i]) then
      begin
        Dedamoji:= Siustuvai[i].Liekanos * Power(PadengiamosZonos(Siustuvai[i])/
                   (Siustuvai[i].SiustuvoKaina + Siustuvai[i].ZemesKaina), Beta);
        Vardiklis:= Vardiklis + Dedamoji;
        if Siustuvas.Identifikatorius = Siustuvai[i].Identifikatorius then
          Skaitiklis:= Dedamoji;
      end;
  if (Skaitiklis = 0) or (Vardiklis = 0) Then
    Result:= 0
  else
    Result:= Skaitiklis / Vardiklis;
end;

function TBevielesPrieigosAlgoritmas.PapildytiSprendini(PartialSolution: TSiustuvaiList;
                                                        Beta: Real): boolean;
var Choose, Sum: real;
    i: integer;
begin
  Randomize;
  Choose:= Random;
  Sum:= 0;
  i:= 0;
  while (Sum <= choose) and (i < Siustuvai.Count) do begin
    Sum:= Sum + PatekimoTikimybe(Siustuvai[i], PartialSolution, Beta);
    if Sum <= choose then
      i:= i + 1;
  end;
  if i < Siustuvai.Count then
  begin
    PartialSolution.Add.Assign(Siustuvai[i]);
    Result:= true;
  end
  else
    Result:= false;
end;

function TBevielesPrieigosAlgoritmas.DaznisSprendiniuose(PartialSolutions: TPartialSolutionsList;
                                                         Siustuvas: TSiustuvaiListItem): Real;
var i, j, Skaitiklis: integer;
begin
  Skaitiklis:= 0;
  for i:= 0 to PartialSolutions.Count - 1 do begin
    for j:= 0 to PartialSolutions[i].Solution.Count - 1 do
      if PartialSolutions[i].Solution[j].Identifikatorius = Siustuvas.Identifikatorius then
      begin
        Skaitiklis:= Skaitiklis + 1;
        break;
      end;
  end;
  if PartialSolutions.Count > 0 then
    Result:= Skaitiklis / PartialSolutions.Count
  else
    Result:= 0;
end;

procedure TBevielesPrieigosAlgoritmas.AtnaujintiFeromonus(PartialSolutions: TPartialSolutionsList;
                                                          Ro: real);
var i: Integer;
begin
  for i:= 0 to Siustuvai.Count -1 do
    Siustuvai[i].Liekanos:= (1 - Ro) * Siustuvai[i].Liekanos + Ro *
                            DaznisSprendiniuose(PartialSolutions, Siustuvai[i]);
end;
```

```
function TBevielesPrieigosAlgoritmas.TinkloKastai(Solution: TSiustuvaiList): real;
var i: integer;
begin
  Result:= 0;
  for i:= 0 to Solution.Count - 1 do
    Result:= Result + Solution[i].SiustuvoKaina + Solution[i].ZemesKaina;
end;

procedure                    TBevielesPrieigosAlgoritmas.AtnaujintiGeriausiaSprendini(PartialSolutions:
TPartialSolutionsList;
                                                                    out Solution: TSiustuvaiList);
var i, j, bestNum: Integer;
    MinPrice, TempMinPrice: Real;
begin
  if Solution = nil then
  begin
    Solution:= TSiustuvaiList.Create;
    MinPrice:= MaxExtended;
  end
  else
    minPrice:= TinkloKastai(Solution);
  bestNum:= -1;
  for i:= 0 to PartialSolutions.Count - 1 do
  begin
    TempMinPrice:= TinkloKastai(PartialSolutions[i].Solution);
    if TempMinPrice < MinPrice then
    begin
      MinPrice:= TempMinPrice;
      bestNum:= i;
    end;
  end;
  if bestNum > -1 then
  begin
    Solution.Assign(PartialSolutions[bestNum].Solution);
    for i:= 0 to Solution.Count - 1 do
    begin
      Solution[i].AptarnaujamosZonos.KopijuotiZonuParametrus(Siustuvai.Zonos);
      for j:= 0 to Solution[i].PadengimoVektorius.Count - 1 do
        if Solution[i].PadengimoVektorius[j].Padengta > 0 then
          Solution[i].AptarnaujamosZonos.Add.Assign(Siustuvai.Zonos[j]);
    end;
  end;
end;

function TBevielesPrieigosAlgoritmas.SkruzdziuKolonijosAlgoritmas(IterCount, AntsCount: Integer;
                                                                  NaudojamaLokaliPaieska: boolean;
                                                                  Beta, Ro: Real;
                                                                  out     Solution:     TSiustuvaiList):
Boolean;
var iter, ant, i: integer;
    PartialSolution: TSiustuvaiList;
    PartialSolutions: TPartialSolutionsList;
    PartialSolutionsITem: TPartialSolutionsListItem;
    papildymoRezultatas, Rasta: boolean;
begin
  PartialSolution:= TSiustuvaiList.Create;
  PartialSolutions:= TPartialSolutionsList.Create;
  Solution:= nil;
  Rasta:= false;
  for iter:= 1 to IterCount do
  begin
    PartialSolutions.Clear;
    for ant:= 0 to AntsCount - 1 do
    begin
      PartialSolution.Clear;
      papildymoRezultatas:= true;
      while (not ArGautasLeistinasSprendinys(PartialSolution)) and
            (papildymoRezultatas) do
        papildymoRezultatas:= PapildytiSprendini(PartialSolution, Beta);
      if papildymoRezultatas then
      begin
        if NaudojamaLokaliPaieska then LokaliPaieska(PartialSolution);
        PartialSolutions.Add.Solution.Assign(PartialSolution);
        Rasta:= true;
      end;
    end;
    AtnaujintiGeriausiaSprendini(PartialSolutions, Solution);
    AtnaujintiFeromonus(PartialSolutions, Ro);
  end;
```

```
  PartialSolution.Free;
  PartialSolution:= nil;
  PartialSolutions.Free;
  PartialSolutions:= nil;
  Result:= Rasta;
end;

end.
```

unit LaidinesPrieigosAlgoritmas;

```
interface

uses
  Zonos, Siustuvai, Math, StociuMetodikosInfo, PadengimoVektorius, Classes,
  DuomenuStrukturos, SysUtils;

type

  TLaidinesPrieigosAlgoritmas = class
  private
    Logas: TStringList;
    Siustuvai: TSiustuvaiList;
    procedure LokaliPaieska(Solution: TSiustuvaiList);
    procedure AtnaujintiFeromonus(PartialSolutions: TPartialSolutionsList;
                                  Ro: real);
    procedure AtnaujintiGeriausiaSprendini(PartialSolutions: TPartialSolutionsList;
                                           out Solution: TSiustuvaiList);
    procedure PerskirstytiAptarnaujamasZonas(PartialSolution: TSiustuvaiList;
                                             Siustuvas: TSiustuvaiListItem);
    function IsmestiSiustuva(Num: Integer; Solution: TSiustuvaiList): boolean;
    function PapildytiSprendini(PartialSolution: TSiustuvaiList;
                                Beta: Real): boolean;
    function SkruzdziuKolonijosAlgoritmas(IterCount, AntsCount: integer;
                                          NaudojamaLokaliPaieska: boolean;
                                          Beta, Ro: Real;
                                          out Solution: TSiustuvaiList): boolean;
    function ArGalimaPasalinti(num: Integer; Solution: TSiustuvaiList): boolean;
    function ArPriklausoSprendiniui(Solution: TSiustuvaiList;
                                    Siustuvas: TSiustuvaiListItem): integer;
    function DaznisSprendiniuose(PartialSolutions: TPartialSolutionsList;
                                 Siustuvas: TSiustuvaiListItem): Real;
    function PatekimoTikimybe(Siustuvas: TSiustuvaiListItem;
                              PartialSolution: TSiustuvaiList; Beta: real): real;
    function ArGautasLeistinasSprendinys(Solution: TSiustuvaiList): boolean;
    function ZonosPadengimoLaipsnis(PartialSolution: TSiustuvaiList;
                                    ZoneIndex: Integer): Integer;
    function ReikiaPerskirstyti(ZoneIndex: Integer;
                                PartialSolution: TSiustuvaiList;
                                Siustuvas: TSiustuvaiListItem): boolean;
    function NaujaiPadengiamosZonos(PartialSolution: TSiustuvaiList;
                                    Siustuvas: TSiustuvaiListItem): Integer;
    function TinkloKastai(Solution: TSiustuvaiList): real;
    function PrijungimoKastai(ZoneIndex: Integer; Siustuvas: TSiustuvaiListItem): Real;
    function SiustuvoKaina(Siustuvas: TSiustuvaiListItem): Real;
  protected

  public
    function Apskaiciuoti(PotencialusSiustuvai: TSiustuvaiList;
                          Parametrai: TMetodikosParamList;
                          Log: TStringList;
                          out Sprendinys: TSiustuvaiList;
                          out Kaina, Padengta: real): boolean;
  end;

implementation

function TLaidinesPrieigosAlgoritmas.ArGalimaPasalinti(num: Integer;
                                                       Solution: TSiustuvaiList): boolean;
var suminisPadengimas: TPadengimoVektorius;
    i, j: Integer;
begin
  suminisPadengimas:= TPadengimoVektorius.Create;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    suminisPadengimas.Add;
  for i:= 0 to Solution.Count - 1 do
```

```
    if i <> num then
      for j:= 0 to Solution[i].PadengimoVektorius.Count - 1 do
        suminisPadengimas[j].Padengta:= suminisPadengimas[j].Padengta +
                                        Solution[i].PadengimoVektorius[j].Padengta;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    if suminisPadengimas[i].Padengta < Siustuvai.Zonos[i].PadengimoLaipsnis then
    begin
      Result:= false;
      suminisPadengimas.Free;
      suminisPadengimas:= nil;
      Exit;
    end;
  Result:= true;
  suminisPadengimas.Free;
  suminisPadengimas:= nil;
end;

function TLaidinesPrieigosAlgoritmas.IsmestiSiustuva(Num: Integer;
                                                     Solution: TSiustuvaiList): boolean;
var i, j: Integer;
    priemimoKastai, minPriemimoKastai, minPriemimoKastaiSum: Real;
    priimancioNum: array of Integer;
begin
  SetLength(priimancioNum, Solution[Num].AptarnaujamosZonos.Count);
  minPriemimoKastaiSum:= 0;
  for i:= 0 to Solution[Num].AptarnaujamosZonos.Count - 1 do
  begin
    minPriemimoKastai:= MaxExtended;
    priimancioNum[i]:= -1;
    for j:= 0 to Solution.Count - 1 do
      if (j <> Num) and
         (Solution[j].PadengimoVektorius[Solution[Num].AptarnaujamosZonos[i].Identifikatorius].Padengta
> 0) then
      begin
        priemimoKastai:=         PrijungimoKastai(Solution[Num].AptarnaujamosZonos[i].Identifikatorius,
Solution[j]);
        if priemimoKastai < minPriemimoKastai then
        begin
          minPriemimoKastai:= priemimoKastai;
          priimancioNum[i]:= j;
        end;
      end;
      minPriemimoKastaiSum:= minPriemimoKastaiSum + minPriemimoKastai;
  end;
  if SiustuvoKaina(Solution[Num]) > minPriemimoKastaiSum then
  begin
    for i:= 0 to Solution[Num].AptarnaujamosZonos.Count - 1 do
      Solution[priimancioNum[i]].AptarnaujamosZonos.Add.Assign(Solution[Num].AptarnaujamosZonos[i]);
    Solution.Delete(Num);
    Result:= true;
  end
  else
    Result:= false;
  PriimancioNum:= nil;
end;

procedure TLaidinesPrieigosAlgoritmas.LokaliPaieska(Solution: TSiustuvaiList);
var i: Integer;
begin
  i:= 0;
  while i < Solution.Count do
    if ArGalimaPasalinti(i, Solution) then
    begin
      if not IsmestiSiustuva(i, Solution) then
        i:= i + 1;
    end
    else
      i:= i + 1;
end;

function TLaidinesPrieigosAlgoritmas.Apskaiciuoti(PotencialusSiustuvai: TSiustuvaiList;
                                                  Parametrai: TMetodikosParamList;
                                                  Log: TStringList;
                                                  out Sprendinys: TSiustuvaiList;
                                                  out Kaina, Padengta: Real): boolean;
var i: integer;
    FormatSet: TFormatSettings;
begin
  Logas:= Log;
  Siustuvai:= PotencialusSiustuvai;
```

```
  for i:= 0 to Siustuvai.Count - 1 do
    Siustuvai[i].Liekanos:= 1 / Siustuvai.Count;
  GetLocaleFormatSettings(1033, FormatSet);
  if SkruzdziuKolonijosAlgoritmas(StrToInt(Parametrai[0].Reiksme),
                                  StrToInt(Parametrai[1].Reiksme),
                                  boolean(StrToInt(Parametrai[4].Reiksme)),
                                  StrToFloat(Parametrai[2].Reiksme, FormatSet),
                                  StrToFloat(Parametrai[3].Reiksme, FormatSet),
                                  Sprendinys) then
  begin
    Sprendinys.KopijuotiSiustuvuParametrus(Siustuvai);
    Kaina:= TinkloKastai(Sprendinys);
    Padengta:= 100;
    Result:= true;
  end
  else
    Result:= false;
  Logas:= nil;
  Siustuvai:= nil;
end;

function TLaidinesPrieigosAlgoritmas.ArPriklausoSprendiniui(Solution: TSiustuvaiList;
                                                            Siustuvas: TSiustuvaiListItem): Integer;
var i: integer;
begin
  result:= 0;
  for i:= 0 to Solution.Count - 1 do
    if Solution[i].Identifikatorius = Siustuvas.Identifikatorius then
    begin
      result:= 1;
      exit;
    end;
end;

function TLaidinesPrieigosAlgoritmas.ArGautasLeistinasSprendinys(Solution: TSiustuvaiList): boolean;
var suminisPadengimas: TPadengimoVektorius;
    i, j: Integer;
begin
  suminisPadengimas:= TPadengimoVektorius.Create;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    suminisPadengimas.Add;
  for i:= 0 to Solution.Count - 1 do
    for j:= 0 to Solution[i].AptarnaujamosZonos.Count - 1 do
      suminisPadengimas[Solution[i].AptarnaujamosZonos[j].Identifikatorius].Padengta:=
       suminisPadengimas[Solution[i].AptarnaujamosZonos[j].Identifikatorius].Padengta + 1;
  for i:= 0 to Siustuvai.Zonos.Count - 1 do
    if suminisPadengimas[i].Padengta < Siustuvai.Zonos[i].PadengimoLaipsnis then
    begin
      Result:= false;
      suminisPadengimas.Free;
      suminisPadengimas:= nil;
      Exit;
    end;
  Result:= true;
  suminisPadengimas.Free;
  suminisPadengimas:= nil;
end;

function TLaidinesPrieigosAlgoritmas.ZonosPadengimoLaipsnis(PartialSolution: TSiustuvaiList;
                                                            ZoneIndex: Integer): Integer;
var i, j: Integer;
begin
  Result:= 0;
  for i:= 0 to PartialSolution.Count - 1 do
    for j:= 0 to PartialSolution[i].AptarnaujamosZonos.Count - 1 do
      if PartialSolution[i].AptarnaujamosZonos[j].Identifikatorius = ZoneIndex then
        Result:= Result + 1;
end;

function TLaidinesPrieigosAlgoritmas.ReikiaPerskirstyti(ZoneIndex: Integer;
                                                        PartialSolution: TSiustuvaiList;
                                                        Siustuvas: TSiustuvaiListItem): boolean;
var i: Integer;
    minKastai: Real;
begin
  Result:= true;
  minKastai:= PrijungimoKastai(ZoneIndex, Siustuvas);
  for i:= 0 to PartialSolution.Count - 1 do
    if PartialSolution[i].PadengimoVektorius[ZoneIndex].Padengta > 0 then
      if PrijungimoKastai(ZoneIndex, PartialSolution[i]) <= minKastai then
```

```
      begin
        Result:= false;
        Exit;
      end;
end;

procedure TLaidinesPrieigosAlgoritmas.PerskirstytiAptarnaujamasZonas(PartialSolution: TSiustuvaiList;
                                                                     Siustuvas: TSiustuvaiListItem);
var i, j, z: Integer;
    Perskirstyta: boolean;
begin
  for z:= 0 to Siustuvas.AptarnaujamosZonos.Count - 1 do
  begin
    i:= 0;
    Perskirstyta:= false;
    while (i < PartialSolution.Count) and (not Perskirstyta) do
    begin
      j:= 0;
      while (j < PartialSolution[i].AptarnaujamosZonos.Count) and (not Perskirstyta) do
      begin
        if PartialSolution[i].AptarnaujamosZonos[j].Identifikatorius =
           Siustuvas.AptarnaujamosZonos[z].Identifikatorius then
        begin
          PartialSolution[i].AptarnaujamosZonos.Delete(j);
          Perskirstyta:= true;
        end;
        j:= j + 1;
      end;
      i:= i + 1;
    end;
  end;
end;

function TLaidinesPrieigosAlgoritmas.PrijungimoKastai(ZoneIndex: Integer;
                                                      Siustuvas: TSiustuvaiListItem): Real;
begin
  Result:= Siustuvai.Zonos[ZoneIndex].VartotojuSkaicius *
            Siustuvas.PrijungimoKastaiKm * sqrt(sqr(Siustuvas.X -
            Siustuvai.Zonos[ZoneIndex].X - Siustuvai.Zonos[ZoneIndex].Ilgis / 2) +
            sqr(Siustuvas.Y - Siustuvai.Zonos[ZoneIndex].Y -
            Siustuvai.Zonos[ZoneIndex].Ilgis / 2))
end;

function TLaidinesPrieigosAlgoritmas.NaujaiPadengiamosZonos(PartialSolution: TSiustuvaiList;
                                                            Siustuvas: TSiustuvaiListItem): Integer;
var i: integer;
    AptarnaujamosZonos: TZonosList;
begin
  AptarnaujamosZonos:= TZonosList.Create;
  AptarnaujamosZonos.KopijuotiZonuParametrus(Siustuvai.Zonos);
  Result:= 0;
  for i:= 0 to Siustuvas.PadengimoVektorius.Count - 1 do
    if (Siustuvas.PadengimoVektorius[i].Padengta > 0) and
       (Siustuvai.Zonos[i].PadengimoLaipsnis > 0) then
      if (ZonosPadengimoLaipsnis(PartialSolution, i) < Siustuvai.Zonos[i].PadengimoLaipsnis) or
         ReikiaPerskirstyti(i, PartialSolution, Siustuvas) then
      begin
        Result:= Result + 1;
        AptarnaujamosZonos.Add.Assign(Siustuvai.Zonos[i]);
      end;
  Siustuvas.AptarnaujamosZonos.Assign(AptarnaujamosZonos);
  AptarnaujamosZonos.Free;
  AptarnaujamosZonos:= nil;
end;

function TLaidinesPrieigosAlgoritmas.SiustuvoKaina(Siustuvas: TSiustuvaiListItem): Real;
var i: Integer;
begin
  Result:= Siustuvas.SiustuvoKaina + Siustuvas.ZemesKaina;
  for i:= 0 to Siustuvas.AptarnaujamosZonos.Count - 1 do
    Result:= Result + PrijungimoKastai(Siustuvas.AptarnaujamosZonos[i].Identifikatorius, Siustuvas);
end;

function TLaidinesPrieigosAlgoritmas.PatekimoTikimybe(Siustuvas: TSiustuvaiListItem;
                                                      PartialSolution: TSiustuvaiList;
                                                      Beta: real): real;
var Vardiklis, Skaitiklis, Dedamoji: Real;
    i, NaujaiPadengiamuSk: Integer;
begin
  Vardiklis:= 0;
```

```
  Skaitiklis:= 0;
  for i:= 0 to Siustuvai.Count - 1 do
    if ArPriklausoSprendiniui(PartialSolution, Siustuvai[i]) = 0 then
    begin
      NaujaiPadengiamuSk:= NaujaiPadengiamosZonos(PartialSolution, Siustuvai[i]);
      if NaujaiPadengiamuSk > 0 then
      begin
        Dedamoji:= Siustuvai[i].Liekanos *
                   Power(NaujaiPadengiamuSk / SiustuvoKaina(Siustuvai[i]), Beta);
        Vardiklis:= Vardiklis + Dedamoji;
        if Siustuvas.Identifikatorius = Siustuvai[i].Identifikatorius then
          Skaitiklis:= Dedamoji;
      end;
    end;
  if (Skaitiklis = 0) or (Vardiklis = 0) Then
    Result:= 0
  else
    Result:= Skaitiklis / Vardiklis;
end;

function TLaidinesPrieigosAlgoritmas.PapildytiSprendini(PartialSolution: TSiustuvaiList;
                                                        Beta: Real): boolean;
var Choose, Sum: real;
    i: integer;
begin
  Randomize;
  Choose:= Random;
  Sum:= 0;
  i:= 0;
  while (Sum <= choose) and (i < Siustuvai.Count) do begin
    Sum:= Sum + PatekimoTikimybe(Siustuvai[i], PartialSolution, Beta);
    if Sum <= choose then
      i:= i + 1;
  end;
  if i < Siustuvai.Count then
  begin
    PerskirstytiAptarnaujamasZonas(PartialSolution, Siustuvai[i]);
    PartialSolution.Add.Assign(Siustuvai[i]);
    Result:= true;
  end
  else
    Result:= false;
end;

function TLaidinesPrieigosAlgoritmas.DaznisSprendiniuose(PartialSolutions: TPartialSolutionsList;
                                                         Siustuvas: TSiustuvaiListItem): Real;
var i, j, Skaitiklis: integer;
begin
  Skaitiklis:= 0;
  for i:= 0 to PartialSolutions.Count - 1 do begin
    for j:= 0 to PartialSolutions[i].Solution.Count - 1 do
      if PartialSolutions[i].Solution[j].Identifikatorius = Siustuvas.Identifikatorius then
      begin
        Skaitiklis:= Skaitiklis + 1;
        break;
      end;
  end;
  if PartialSolutions.Count > 0 then
    Result:= Skaitiklis / PartialSolutions.Count
  else
    Result:= 0;
end;

procedure TLaidinesPrieigosAlgoritmas.AtnaujintiFeromonus(PartialSolutions: TPartialSolutionsList;
                                                          Ro: real);
var i: Integer;
begin
  for i:= 0 to Siustuvai.Count -1 do
    Siustuvai[i].Liekanos:= (1 - Ro) * Siustuvai[i].Liekanos + Ro *
                            DaznisSprendiniuose(PartialSolutions, Siustuvai[i]);
end;

function TLaidinesPrieigosAlgoritmas.TinkloKastai(Solution: TSiustuvaiList): real;
var i: integer;
begin
  Result:= 0;
  for i:= 0 to Solution.Count - 1 do
    Result:= Result + SiustuvoKaina(Solution[i]);
end;
```

```
procedure                    TLaidinesPrieigosAlgoritmas.AtnaujintiGeriausiaSprendini(PartialSolutions:
TPartialSolutionsList;
                                                                    out Solution: TSiustuvaiList);
var i, j, bestNum: Integer;
    MinPrice, TempMinPrice: Real;
begin
  if Solution = nil then
  begin
    Solution:= TSiustuvaiList.Create;
    MinPrice:= MaxExtended;
  end
  else
    minPrice:= TinkloKastai(Solution);
  bestNum:= -1;
  for i:= 0 to PartialSolutions.Count - 1 do
  begin
    TempMinPrice:= TinkloKastai(PartialSolutions[i].Solution);
    if TempMinPrice < MinPrice then
    begin
      MinPrice:= TempMinPrice;
      bestNum:= i;
    end;
  end;
  if bestNum > -1 then
  begin
    Solution.Assign(PartialSolutions[bestNum].Solution);
    for i:= 0 to Solution.Count - 1 do
      for j:= 0 to Solution[i].AptarnaujamosZonos.Count - 1 do
        Solution[i].AptarnaujamosZonos[j].PrijungimoKastai:=
        PrijungimoKastai(Solution[i].AptarnaujamosZonos[j].Identifikatorius, Solution[i]);
  end;
end;

function TLaidinesPrieigosAlgoritmas.SkruzdziuKolonijosAlgoritmas(IterCount, AntsCount: Integer;
                                                                  NaudojamaLokaliPaieska: boolean;
                                                                  Beta, Ro: Real;
                                                                  out     Solution:     TSiustuvaiList):
Boolean;
var iter, ant, i: integer;
    PartialSolution: TSiustuvaiList;
    PartialSolutions: TPartialSolutionsList;
    papildymoRezultatas, Rasta: boolean;
begin
  PartialSolution:= TSiustuvaiList.Create;
  PartialSolutions:= TPartialSolutionsList.Create;
  Solution:= nil;
  Rasta:= false;
  for iter:= 1 to IterCount do
  begin
    PartialSolutions.Clear;
    for ant:= 0 to AntsCount - 1 do
    begin
      PartialSolution.Clear;
      papildymoRezultatas:= true;
      while (not ArGautasLeistinasSprendinys(PartialSolution)) and
            (papildymoRezultatas) do
        papildymoRezultatas:= PapildytiSprendini(PartialSolution, Beta);
      if papildymoRezultatas then
      begin
        if NaudojamaLokaliPaieska then LokaliPaieska(PartialSolution);
        PartialSolutions.Add.Solution.Assign(PartialSolution);
        Rasta:= true;
      end;
    end;
    AtnaujintiGeriausiaSprendini(PartialSolutions, Solution);
    AtnaujintiFeromonus(PartialSolutions, Ro);
  end;
  PartialSolution.Free;
  PartialSolution:= nil;
  PartialSolutions.Free;
  PartialSolutions:= nil;
  Result:= Rasta;
end;

end.
```

unit Algoritmas;

```
interface
```

```
uses
  Windows, Dialogs, Series, Classes, SysUtils, StdCtrls, Variants, Duomenys, ApribCol,
  ParamCol, ParamRik, Siustuvai, StociuMetodikosInfo;

type
  TPazeidimuVektorius = array of Integer;

  TAlgoritmas = class
  private
    FDuomenys: TDuomenys; //Visi optimizavimo algortmui reikaligi duomenys
    FEkranoLogas: TStrings; //Algoritmo darbo isvedimui ekrane
    FFileLog: TStrings; //Algoritmo darbo isvedimui faile
    FKastuGrafikas: TBarSeries; //Kastu grafikas
    FStociuMetodikos: TStociuMetodikosInfo;
    ////////////////////////////////////////////////////////////////////////////
    ////////////////Funkcijos, skirtos isvesti algoritmo darbui/////////////////
    ////////////////////////////////////////////////////////////////////////////
    procedure IsvestiBylosAntraste; //Isvedama antraste i faila
    procedure IsvestiReiksmes(KKastai: Variant);  //Isvedamos reiksmes i faila
    procedure IsvestiRodikli(Rod: TApribListItem); //Isveda parametru nurodyta
                                                   //kokybes rodikli
    procedure AtvaizduotiKastus(Iter: integer; KKastai: Variant); //Atvaizduoja
                                                                  //pokyti kastu
    procedure AtvaizduotiApribojimus; //Atvaizduoja apribojimu pokyti
    procedure AtvaizduotiStociuIsdestyma;
    procedure IsvestiLinija; //Isvedama skirianti linija
    ////////////////////////////////////////////////////////////////////////////
    //Funkcijos, apskaiciuojancios rodiklius ir atliekancios reiksmiu keitimus//
    ////////////////////////////////////////////////////////////////////////////
    procedure ApskaiciuotiApribojimuReiksmes; //Kontroliuoja kokybes rodikliu
                                              //apskaiciavimo procedura
    function TikrintiApribojimuPazeidimus: TPazeidimuVektorius; //Formuoja pazeidimu
                                                                //vektoriu
    function TaisytiPazeidima(Num, KN: integer; //Bando pataisyti atsiradusi pazeidima
                              var PradiniaiPazeidimai: TPazeidimuVektorius): boolean;
    function PriskirtiNauja(var Pasirinkta: integer; Kintamasis, Apribojimas,
                            Kryptis: Integer): boolean; //Pakeicia kintamojo reiksme
    ////////////////////////////////////////////////////////////////////////////
    ////////////Funkcijos, atliekancios leistinojo spredinio paieska////////////
    ////////////////////////////////////////////////////////////////////////////
    procedure FiksuotiPradiniRinkini(ParinkimoBudas: integer); //Fiksuoja pradini
                                                              //reiksmiu rinkini
    function ArNeatsiradoNaujuPazeidimu(PradiniaiPazeidimai,             //Patikrina ar
                                        Pazeidimai: TPazeidimuVektorius; //neatsiranda
                                        Num: integer): boolean;          //pazeidimu
    function LeistinojoSprendinioPaieska: boolean; //Kontroliuoja leistinojo
                                                   //sprendinio paieska
    ////////////////////////////////////////////////////////////////////////////
    ////////////////////Funkcijos, atliekancios optimizavima////////////////////
    ////////////////////////////////////////////////////////////////////////////
    function ArGalimaTaisyti(Kint: TParamListItem): boolean; //Patikrina ar nuo
                                                            //uzduoto kintamojo
                                                            //nepriklauso kastu
                                                            //funkcija
    function ArNeatsiradoPazeidimu(KN: integer; Pazeidimai: TPazeidimuVektorius): boolean;
                                  //Patikrina ar neatsiranda pazeidimu keiciant reiksme
    procedure OptimalausSprendinioPaieska; //Kontroliuoja optimizavimmo algoritma
    ////////////////////////////////////////////////////////////////////////////
    procedure IsdestytiStotis;
  protected

  public
    constructor Create(Duom: TDuomenys; StociuIsdestymoMetodika: TStociuMetodikosInfo;
                       EkranoLogas: TStrings; KGrafikas: TBarSeries);
    destructor Destroy; override;
    function Optimizuoti(ParinkimoBudas, Rezimas: integer): boolean;
  end;

implementation
////////////////////////////////////////////////////////////////////////////////
//////////////////Funkcijos, skirtos isvesti algoritmo darbui///////////////////
////////////////////////////////////////////////////////////////////////////////
procedure TAlgoritmas.IsvestiBylosAntraste; //Isvedama antraste i faila
var i: integer;
    Temp: string;
begin
  Temp:= '';
  with FDuomenys do
  begin
```

```
    for i:= 0 to Kintamieji.Count - 1 do
      Temp:= Temp + Kintamieji[i].Pav + ListSeparator;
    for i:= 0 to Apribojimai.Count - 1 do
      Temp:= Temp + Apribojimai[i].Pav + ListSeparator;
    Temp:= Temp + 'Kastai';
  end;
  FFileLog.Add(Temp);
end;
////////////////////////////////////////////////////////////////////////////
procedure TAlgoritmas.IsvestiReiksmes(KKastai: Variant);  //Isvedamos reiksmes i faila
var i: integer;
    Temp: string;
begin
  Temp:= '';
  with FDuomenys do
  begin
    for i:= 0 to Kintamieji.Count - 1 do
      Temp:=    Temp    +    VarToStr(Kintamieji[i].ReiksList[Kintamieji[i].Pasirinkta].Reiksme)    +
ListSeparator;
    if Apribojimai.Count > 1 then
      for i:= 0 to Apribojimai.Count - 1 do
        Temp:= Temp + VarToStr(Apribojimai[i].Apskaiciuota) + ListSeparator;
    if KKastai > 0 then Temp:= Temp + VarToStr(KKastai);
  end;
  FFileLog.Add(Temp);
end;
////////////////////////////////////////////////////////////////////////////
procedure TAlgoritmas.IsvestiRodikli(Rod: TApribListItem); //Isveda parametru
                                                           //nurodyta kokybes rodikli
begin
  FEkranoLogas.Add('Apribojimo pavadinimas: ' + Rod.Pav);
  FEkranoLogas.Add('Min reiksme: ' + VarToStr(Rod.MinReiksme));
  FEkranoLogas.Add('Max reiksme: ' + VarToStr(Rod.MaxReiksme));
  FEkranoLogas.Add('Apskaiciuota: ' + VarToStr(Rod.Apskaiciuota));
  IsvestiLinija;
end;
////////////////////////////////////////////////////////////////////////////
procedure TAlgoritmas.AtvaizduotiKastus(Iter: integer; KKastai: Variant); //Atvaizduoja pokyti kastu
begin
  FEkranoLogas.Add('Kastai: ' + FloatToStrF(KKastai, ffFixed, 18, 2));
  IsvestiLinija;
  IsvestiReiksmes(KKastai);
  FKastuGrafikas.AddXY(Iter, KKastai);
  FKastuGrafikas.Repaint;
end;
////////////////////////////////////////////////////////////////////////////
procedure TAlgoritmas.AtvaizduotiStociuIsdestyma; //Atvaizduoja stociu isdestymo rezultatus
begin
  if FStociuMetodikos.SprendinysSukonstruotas then
    FEkranoLogas.Add('Teritorija padengiti pavyko')
  else
     FEkranoLogas.Add('Teritorijos padengti nepavyko');
  IsvestiLinija;
end;
////////////////////////////////////////////////////////////////////////////
procedure TAlgoritmas.AtvaizduotiApribojimus; //Atvaizduoja apribojimu pokyti
begin
  IsvestiReiksmes(-1);
end;
////////////////////////////////////////////////////////////////////////////
procedure TAlgoritmas.IsvestiLinija;
begin
  FEkranoLogas.Add('-----------------------------------------------------');
end;
////////////////////////////////////////////////////////////////////////////
////Funkcijos, apskaiciuojancios rodiklius ir atliekancios reiksmiu keitimus////
////////////////////////////////////////////////////////////////////////////
procedure TAlgoritmas.ApskaiciuotiApribojimuReiksmes; //Kontroliuoja kokybes rodikliu
                                                      //apskaiciavimo procedura
var i: integer;
begin
  for i:= 0 to FDuomenys.Apribojimai.Count - 1 do
  begin
    FDuomenys.Apribojimai[i].Apskaiciuoti(FDuomenys.Kintamieji);
    IsvestiRodikli(FDuomenys.Apribojimai[i]);
  end;
end;
////////////////////////////////////////////////////////////////////////////
function TAlgoritmas.TikrintiApribojimuPazeidimus: TPazeidimuVektorius; //Formuoja
                                                                        //pazeidimu
```

```
                                                                          //vektoriu
var i: integer;
    Vektorius: TPazeidimuVektorius;
begin
  SetLength(Vektorius, FDuomenys.Apribojimai.Count + 1);
  for i:= 0 to Length(Vektorius) - 2 do
    if (FDuomenys.Apribojimai[i].Apskaiciuota < FDuomenys.Apribojimai[i].MinReiksme) then
      Vektorius[i]:= -1
    else if (FDuomenys.Apribojimai[i].Apskaiciuota > FDuomenys.Apribojimai[i].MaxReiksme) then
           Vektorius[i]:= 1
         else Vektorius[i]:= 0;
  if not FStociuMetodikos.SprendinysSukonstruotas then
    Vektorius[i]:= 1
  else
    Vektorius[i]:= 0;
  Result:= Vektorius;
end;
////////////////////////////////////////////////////////////////////////////
function TAlgoritmas.TaisytiPazeidima(Num, KN: integer; //Bando pataisyti atsiradusi pazeidima
                                      var PradiniaiPazeidimai: TPazeidimuVektorius): boolean;
var i, SenaReiksme, Reiksme: integer;
    Kint: TParamListItem;
    Priskirta, Galima, Baigta: boolean;
    Pazeidimai: TPazeidimuVektorius;
    Rikiuote: TParamRikList;
begin
  Baigta:= false;
  if Num < FDuomenys.Apribojimai.Count then
    Rikiuote:= FDuomenys.DuomenuRikiuote[Num].ParamRikList
  else
    Rikiuote:= FStociuMetodikos.IsdestymoRikiuote;

    for i:= 0 to Rikiuote.Count - 1 do
    begin
      Kint:= FDuomenys.Kintamieji[Rikiuote[i].Num];
      if KN > -1 then
        if not ArGalimaTaisyti(Kint) then Continue;
      FEkranoLogas.Add(Kint.Pav + ' reiksmiu perrinkimas');
      IsvestiLinija;
      repeat
        SenaReiksme:= Kint.Pasirinkta;
        Reiksme:= Kint.Pasirinkta;
        Priskirta:= PriskirtiNauja(Reiksme, i, Num, PradiniaiPazeidimai[Num]);
        if Priskirta then
        begin
          Kint.Pasirinkta:= Reiksme;
          FEkranoLogas.Add('Sena reiksme: ' + Kint.ReiksList[SenaReiksme].Reiksme);
          FEkranoLogas.Add('Nauja reiksme: ' + Kint.ReiksList[Reiksme].Reiksme);
          IsvestiLinija;
          IsdestytiStotis;
          ApskaiciuotiApribojimuReiksmes;
          AtvaizduotiApribojimus;
          AtvaizduotiStociuIsdestyma;
          Pazeidimai:= TikrintiApribojimuPazeidimus;
          Galima:= false;
          if ArNeatsiradoNaujuPazeidimu(PradiniaiPazeidimai, Pazeidimai, Num) then
          begin
            Galima:= true;
            if Pazeidimai[Num] = 0 then Baigta:= true;
            PradiniaiPazeidimai:= Pazeidimai;
          end
          else
          begin
            Kint.Pasirinkta:= SenaReiksme;
            FEkranoLogas.Add('Reiksmes pakeitimo atsisakome');
            IsvestiLinija;
            FStociuMetodikos.Kastai:= FStociuMetodikos.SeniKastai;
            FStociuMetodikos.PasirinktiSiustuvai.Assign(FStociuMetodikos.SeniPasirinktiSiustuvai);
            FStociuMetodikos.PadengimoProc:= FStociuMetodikos.SenasPadengimoProc
          end;
        end;
      until (not Priskirta) or (not Galima) or (Baigta);
      if Baigta then
      begin
        Result:= true;
        Exit;
      end;
    end;
  Result:= false;
end;
```

```
////////////////////////////////////////////////////////////////////////////
function TAlgoritmas.PriskirtiNauja(var Pasirinkta: integer; Kintamasis, Apribojimas,
                                    Kryptis: integer): boolean; //Pakeicia kintamojo reiksme
var i: integer;
    Param: TParamRikListItem;
begin
  if Apribojimas < FDuomenys.Apribojimai.Count then
    if Apribojimas >= 0 then
      Param:= FDuomenys.DuomenuRikiuote[Apribojimas].ParamRikList[Kintamasis]
    else
      Param:= FDuomenys.KastuRikiuote[Kintamasis]
  else
    Param:= FStociuMetodikos.IsdestymoRikiuote[Kintamasis];
  with Param do
  begin
    for i:= 0 to ReiksRikList.Count - 1 do
      if Pasirinkta = ReiksRikList[i].Num then
      begin
        case Kryptis of
          -1: if i > 0 then
              begin
                Pasirinkta:= ReiksRikList[i - 1].Num;
                Result:= true;
              end
              else Result:= false;
          1: if i < ReiksRikList.Count - 1 then
             begin
               Pasirinkta:= ReiksRikList[i + 1].Num;
               Result:= true;
             end
             else Result:= false;
        end;
        Exit;
      end;
   end;
end;
////////////////////////////////////////////////////////////////////////////
//////////////Funkcijos, atliekancios leistinojo spredinio paieska//////////
////////////////////////////////////////////////////////////////////////////
procedure TAlgoritmas.FiksuotiPradiniRinkini(ParinkimoBudas: integer); //Fiksuoja pradini
                                                                       //reiksmiu rinkini
var i: integer;
begin
  FEkranoLogas.Add('Kintamasis; Reiksme');
  case ParinkimoBudas of
    0:  //Parekame atsitiktinai
    begin
      Randomize;
      with FDuomenys do
        for i:= 0 to Kintamieji.Count - 1 do
        begin
          if i > 0 then
            Kintamieji[i].Pasirinkta:= Random(Kintamieji[i].ReiksList.Count)
          else
            Kintamieji[i].Pasirinkta:=
FStociuMetodikos.IsdestymoRikiuote[0].ReiksRikList[FStociuMetodikos.IsdestymoRikiuote[0].ReiksRikList.C
ount - 1].Num;
          FEkranoLogas.Add(Kintamieji[i].Pav             +             ';             '             +
Kintamieji[i].ReiksList[Kintamieji[i].Pasirinkta].Reiksme);
        end;
    end;
    1:  //Parenkame pirma is saraso
    with FDuomenys do
      for i:= 0 to Kintamieji.Count - 1 do
        if Kintamieji[i].ReiksList.Count > 0 then
        begin
          if i > 0 then
            Kintamieji[i].Pasirinkta:= 0
          else
            Kintamieji[i].Pasirinkta:=
FStociuMetodikos.IsdestymoRikiuote[0].ReiksRikList[FStociuMetodikos.IsdestymoRikiuote[0].ReiksRikList.C
ount - 1].Num;
          FEkranoLogas.Add(Kintamieji[i].Pav             +             ';             '             +
Kintamieji[i].ReiksList[Kintamieji[i].Pasirinkta].Reiksme);
        end;
  end;
  IsvestiLinija;
end;
////////////////////////////////////////////////////////////////////////////
function TAlgoritmas.ArNeatsiradoNaujuPazeidimu(PradiniaiPazeidimai,
```

```
                                                Pazeidimai: TPazeidimuVektorius;
                                                Num: integer): boolean;
                                                //Patikrina ar neatsiranda pazeidimu
var i: integer;
begin
  for i:= 0 to Length(Pazeidimai) - 1 do
    if i = Num then
    begin
      if PradiniaiPazeidimai[i] <> Pazeidimai[i] then
        if Pazeidimai[i] <> 0 then
        begin
          Result:= false;
          Exit;
        end;
    end
    else
      if PradiniaiPazeidimai[i] <> Pazeidimai[i] then
        begin
          Result:= false;
          Exit;
        end;
  Result:= true;
end;
////////////////////////////////////////////////////////////////////////////
function TAlgoritmas.LeistinojoSprendinioPaieska: boolean; //Kontroliuoja leistinojo
                                                           //sprendinio paieska
var PradiniaiPazeidimai: TPazeidimuVektorius;
    Str: string;
    i: integer;
begin
  IsdestytiStotis;
  ApskaiciuotiApribojimuReiksmes;
  AtvaizduotiApribojimus;
  AtvaizduotiStociuIsdestyma;
  FEkranoLogas.Add('Pazeidimu vektorius');
  IsvestiLinija;
  PradiniaiPazeidimai:= TikrintiApribojimuPazeidimus;
  Str:= '(';
  for i:= 0 to Length(PradiniaiPazeidimai) - 1 do
  begin
    Str:= Str + IntToStr(PradiniaiPazeidimai[i]);
    if i < Length(PradiniaiPazeidimai) - 1 then Str:= Str + ', ';
  end;
  Str:= Str + ')';
  FEkranoLogas.Add(Str);
  IsvestiLinija;
  for i:= 0 to Length(PradiniaiPazeidimai) - 1 do
    if PradiniaiPazeidimai[i] <> 0 then
    begin
      FEkranoLogas.Add(IntToStr(i + 1) + '-tojo pazeidimo taisymas');
      IsvestiLinija;
      if(not TaisytiPazeidima(i, -1, PradiniaiPazeidimai)) then
      begin
        Result:= false;
        Exit;
      end;
  end;
  Result:= true;
end;
////////////////////////////////////////////////////////////////////////////
//////////////////////Funkcijos, atliekancios optimizavima//////////////////
////////////////////////////////////////////////////////////////////////////
function TAlgoritmas.ArGalimaTaisyti(Kint: TParamListItem): boolean; //Patikrina ar nuo
                                                                     //uzduoto kintamojo
                                                                     //nepriklauso kastu
                                                                     //funkcija
var i: integer;
begin
  for i:= 0 to FDuomenys.KastuRikiuote.Count - 1 do
    if FDuomenys.KastuRikiuote[i].Num = Kint.Index then
    begin
      Result:= false;
      Exit;
    end;
  Result:= true;
end;
////////////////////////////////////////////////////////////////////////////
function TAlgoritmas.ArNeatsiradoPazeidimu(KN: integer; Pazeidimai: TPazeidimuVektorius): boolean;
                                          //Patikrina ar neatsiranda pazeidimu keiciant reiksme
var i: integer;
```

```
begin
  for i:= 0 to Length(Pazeidimai) - 1 do
    if Pazeidimai[i] <> 0 then //Badom taisyti pazeidima
    begin
      FEkranoLogas.Add('Pazeidimu taisymo algoritmas ' + IntToStr(i + 1) + '-ajam pazeidimui');
      IsvestiLinija;
      if not TaisytiPazeidima(i, KN, Pazeidimai) then
      begin
        Result:= false;
        Exit;
      end;
    end;
  Result:= true;
end;
////////////////////////////////////////////////////////////////////////////
procedure TAlgoritmas.OptimalausSprendinioPaieska; //Kontroliuoja optimizavimmo algoritma
var i, Iter, SenaReiksme, Reiksme: integer;
    Kint: TParamListItem;
    Priskirta, Galima, Keista: boolean;
    Pazeidimai: TPazeidimuVektorius;
begin
  Iter:= 1;
  AtvaizduotiKastus(0, FStociuMetodikos.Kastai);
  with FDuomenys do
  repeat
    Keista:= false;
    for i:= 0 to KastuRikiuote.Count - 1 do
    begin
      Kint:= Kintamieji[KastuRikiuote[i].Num];
      FEkranoLogas.Add(Kint.Pav + ' reiksmiu perrinkimas');
      IsvestiLinija;
      repeat
        SenaReiksme:= Kint.Pasirinkta;
        Reiksme:= Kint.Pasirinkta;
        Priskirta:= PriskirtiNauja(Reiksme, i, -1, 1);
        if Priskirta then
        begin
          Kint.Pasirinkta:= Reiksme;
          FEkranoLogas.Add('Sena reiksme: ' + Kint.ReiksList[SenaReiksme].Reiksme);
          FEkranoLogas.Add('Nauja reiksme: ' + Kint.ReiksList[Reiksme].Reiksme);
          IsvestiLinija;
          IsdestytiStotis;
          ApskaiciuotiApribojimuReiksmes;
          AtvaizduotiApribojimus;
          AtvaizduotiStociuIsdestyma;
          Pazeidimai:= TikrintiApribojimuPazeidimus;
          Galima:= false;
          if (FStociuMetodikos.Kastai < FStociuMetodikos.SeniKastai) and
              ArNeatsiradoPazeidimu(i, Pazeidimai) then
          begin
            Galima:= true;
            Keista:= true;
            AtvaizduotiKastus(Iter, FStociuMetodikos.Kastai);
            Iter:= Iter + 1;
          end
          else
          begin
            Kint.Pasirinkta:= SenaReiksme;
            FEkranoLogas.Add('Reiksmes pakeitimo atsisakome');
            IsvestiLinija;
            FStociuMetodikos.Kastai:= FStociuMetodikos.SeniKastai;
            FStociuMetodikos.PasirinktiSiustuvai.Assign(FStociuMetodikos.SeniPasirinktiSiustuvai);
            FStociuMetodikos.PadengimoProc:= FStociuMetodikos.SenasPadengimoProc
          end;
        end;
      until (not Priskirta) or (not Galima);
    end;
  until not Keista;
end;
////////////////////////////////////////////////////////////////////////////
constructor TAlgoritmas.Create(Duom: TDuomenys; StociuIsdestymoMetodika: TStociuMetodikosInfo;
                               EkranoLogas: TStrings; KGrafikas: TBarSeries);
begin
  FDuomenys:= Duom;
  FEkranoLogas:= EkranoLogas;
  FFileLog:= TStringList.Create;
  FKastuGrafikas:= KGrafikas;
  FStociuMetodikos:= StociuIsdestymoMetodika;
end;
////////////////////////////////////////////////////////////////////////////
```

```
destructor TAlgoritmas.Destroy;
begin
  FFileLog.Free;
  FFileLog:= nil;
  FDuomenys:= nil;
  FEkranoLogas:= nil;
  FKastuGrafikas:= nil;
  FStociuMetodikos:= nil;
  inherited Destroy;
end;
////////////////////////////////////////////////////////////////////////////
function TAlgoritmas.Optimizuoti(ParinkimoBudas, Rezimas: integer): boolean;
begin
  FDuomenys.Kastai.Apskaiciuota:= '';
  IsvestiBylosAntraste;
  IsvestiLinija;
  FEkranoLogas.Add('Fiksuotas pradiniu reiksmiu rinkinys');
  IsvestiLinija;
  FiksuotiPradiniRinkini(ParinkimoBudas);
  FEkranoLogas.Add('Leistinojo sprendinio paieska');
  IsvestiLinija;
  if LeistinojoSprendinioPaieska then
  begin
    FEkranoLogas.Add('Optimalaus sprendinio paieska');
    IsvestiLinija;
    OptimalausSprendinioPaieska;
    FEkranoLogas.Add('Darbo pabaiga');
    Result:= true;
  end
  else Result:= false;
  FFileLog.SaveToFile('Algoritmas.csv');
end;
////////////////////////////////////////////////////////////////////////////
procedure TAlgoritmas.IsdestytiStotis;
begin
  FStociuMetodikos.Apskaiciuoti(FDuomenys, FEkranoLogas);
end;

end.
```