VYTAUTO DIDŽIOJO UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS INSTITUTAS

Gražvydas Felinskas

EURISTINIŲ METODŲ TYRIMAS IR TAIKYMAS RIBOTŲ IŠTEKLIŲ TVARKARAŠČIAMS OPTIMIZUOTI

Daktaro disertacija

Fiziniai mokslai (P 000)
Informatika (09 P)
Informatika, sistemų teorija (P 175)

Vilnius 2007

Disertacija rengta 2002-2007 metais Matematikos ir informatikos institute

Mokslinis vadovas:
Prof. habil. dr. Leonidas Sakalauskas (Matematikos ir informatikos institutas, fiziniai mokslai, informatika – 09 P)

Turinys:

Žymenys

n – užduočių skaičius.

$J = \{0,1,...,n,n+1\}$ – užduočių indeksų aibė.

$p_j, j \in J$ – užduočių trukmės.

$C = \{ (i, j) \mid i \text{ vykdomas prieš } j \}$ – nuoseklumo sąryšiai aibėje J, porų aibė arba matrica.

m – išteklių rūšių skaičius.

$K = \{1,...,m\}$ – išteklių indeksų aibė.

$R_k > 0, k \in K$ – išskiriama atskiros rūšies išteklių apimtis.

$r_{jk} \geq 0$ – k-tųjų išteklių apimtis (sąnaudos), kurios reikia j-ajai užduočiai atlikti.

$s_j \geq 0$ – j-osios užduoties atlikimo pradžios momentas.

$c_j = s_j + p_j$ – j-osios užduoties pabaigos momentas.

$A(t) = \{ j \in J \mid s_j \leq t < c_j \}$ – užduočių, atliekamų laiko momentu t aibė.

$T(S) = c_{n+1}$ – projekto užbaigimo laikas.

λ_{ij} – minimalus atstumas (laukimo laikas) tarp užduoties i baigimo momento ir užduoties j pradžios momento.

ES_j – ankstyvasis užduoties j pradžios laikas. EF_j – ankstyvasis užduoties j pabaigos laikas.

LS_j – vėlyvasis užduoties j pradžios laikas. LF_j – vėlyvasis užduoties j pabaigos laikas.

$SS_{ij} \geq 0$ – minimalus laiko tarpas tarp užduočių i ir j pradžių (start-to-start),

$FF_{ij} \geq 0$ – minimalus laiko tarpas tarp užduočių i ir j pabaigų (finish-to-finish),

$FS_{ij} \geq 0$ – minimalus laiko tarpas tarp užduoties i pabaigos ir užduoties j pradžios (finish-to-start),

$SF_{ij} \geq 0$ – minimalus laiko tarpas tarp užduoties i pradžios ir užduoties j pabaigos (start-to-finish).

$b = (0, b_1, b_2, ..., b_n, b_{n+1})$ – užduočių numerių pirmumo sąrašas.

$KL(i)$ – užduoties i kritinis laikas.

$D = \{d_{ij}, 1 \leq i, j \leq n\}$ – užduočių nuoseklumo matrica.

$G = \{g_{ij}, 1 \leq i, j \leq n\}$ – pilnoji užduočių nuoseklumo matrica.

ρ_k – aplinkos spindulys k-ajame žingsnyje (iteracijoje).

t_k – temperatūra k-ajame žingsnyje (iteracijoje).

P_{mut} – mutavimo tikimybė.

KiekChrom – chromosomų skaičius populiacijoje.

MAX_IT – maksimalus leistinas iteracijų skaičius.

$P|\#\leq k|C_{max}$ – užduočių paskirstymo keliems lygiagretiems procesoriams su apribojimais uždavinys.

Terminų žodynėlis ir santrumpos

Procesoriumi (mašina) galime vadinti bet kokį vykdytoją, kuris atlieka ar apdoroja paskirtąją užduotį. Tokiu procesoriumi gali būti asmuo, agregatas arba bet koks įtaisas, apdorojantis paskirtas užduotis. Atskiri procesoriai vykdantys tą pačią užduotį gali skirtis užduočių vykdymo intensyvumu (tempu, greičiu).

RITSU (RCPSP) – Ribotų išteklių tvarkaraščių sudarymo uždavinys (Resource Constrained Project Scheduling Problem).

PSPLib – Tvarkaraščių sudarymo uždavinių testinių pavyzdžių biblioteka (Project Scheduling Problem Library).

MA metodas (SA) – Modeliuojamojo atkaitinimo metodas (Simulated annealing).

GA – Genetinis algoritmas (Genetic algorithm).

KKM (CPM) – Kritinių kelių metodas (Critical Path method).

FIFO – Taisyklė „pirmas atėjo – pirmas išėjo“ (First In First Out).

Euristika – kokia nors taisyklė, principas ar metodika, nusakanti, kaip ir kokį veiksmą reikia atlikti, kad surasti kokį nors sprendinį ar priimti kokį nors sprendimą.

Metaeuristika – sudėtingesnės strategijos, kurios gali valdyti kitas euristikas, tam, kad rasti ypač sudėtingų uždavinių sprendinius. Metaeuristikų pavyzdžiai – paieška su draudimais, modeliuojamojo atkaitinimo metodas, genetiniai algoritmai ir kt.

Makespan – projekto (tam tikro užduočių kiekio) vykdymo trukmė.

Pirmumo sąrašas (priority list) – užduočių numerių vektorius, rodantis, kokia tvarka užduotys bus pateikiamos tvarkaraščio dekodavimo procedūrai. Dažniausiai šio sąrašo komponentės surašytos taip, kad tenkintų nuoseklumo sąryšius.

HP – Hewlett Packard (kompanija).

TS, Tabu Search – paieška su draudimais.

SS, Scatter Search – išbarstytoji paieška.

M-C, Monte-Carlo method – Monte Karlo metodas.

VNS, Variable Neighborhood Search – Kintamos aplinkos paieška.

Shift – užduoties perkėlimo operacija.

Swap – užduočių sukeitimo operacija.

Knapsack problem – kuprinės uždavinys.

1 skyrius. Įvadas

1.1 Tyrimų sritis

Inžinerinio projektavimo ir vadybos uždaviniuose dažnai susiduriama su įvairiomis darbų arba užduočių tvarkaraščių sudarymo bei kalendorinio planavimo problemomis. Tokių uždavinių pradiniai duomenys yra informacija apie aibę tam tikrų užduočių, susijusių nuoseklumo sąryšiais. Paprastai laikoma, kad užduotis per tam tikrą laiką atlieka tam skirti procesoriai (vykdytojai, mašinos), naudojantys tam reikalingus išteklius. Ištekliai bei laikas, reikalingas projektui įgyvendinti, gali būti riboti. Siekiant minimizuoti tam tikrus kriterijus, reikia rasti tokį užduočių atlikimo tvarkaraštį, kuris tenkintų nuoseklumo sąryšius, procesorių pajėgumą bei išteklių ribojimus. Didžioji dalis tvarkaraščių sudarymo uždavinių yra NP sudėtingumo. Jiems spręsti dažnai yra taikomi euristiniai metodai. Šio darbo tyrimo sritis yra užduočių tvarkaraščių planavimo ir optimizavimo problemų tyrimas.

1.2 Problemos aktualumas

Optimalių tvarkaraščių radimas yra labai aktuali problema įvairiose veiklos srityse: logistikoje, vadyboje, versle, projektavime, ekonomikoje ir kitose srityse. Leistiną tvarkaraščio sprendinį, kuris tenkina visus ribojimus, rasti klasikiniais metodais, palyginti, nesunku. Pavyzdys – kritinių kelių, mažėjančių trukmių ir kiti panašūs algoritmai. Tačiau toks leistinas tvarkaraštis nebūtinai bus optimalus. Skirtumas tarp minėtais būdais rasto leistinojo ir optimalaus tvarkaraščių gali siekti kelis ar net keliasdešimt procentų. Šis skirtumas įvairiose verslo srityse rinkos konkurencijos sąlygomis bei racionalaus išteklių panaudojimo požiūriu gali būti net labai ženklus.

Optimalių tvarkaraščių sudarymas bendru atveju yra NP sudėtingumo problema, kurią galima išspręsti pilnu binariniu perrinkimu arba pritaikius metodus, besiremiančius šakų ir ribų metodo idėjomis. Tačiau tokius metodus sunku arba neįmanoma realizuoti realiu laiku praktikoje sutinkamiems uždaviniams spręsti. Todėl pastaruoju metu tvarkaraščiams optimizuoti yra taikomi euristiniai optimizavimo metodai, besiremiantys įvairiomis sprendinio paieškos paradigmomis, kurios dažnai kuriamos vadovaujantis analogijomis su gamta, pritaikant dirbtinio intelekto technologijas ir pan. Kompleksinių procesų, apimančių operacijas su keliomis euristikomis, nagrinėjimas yra priskiriamas metaeuristikos sričiai. Šie procesai yra modifikuojami ir derinami tam, kad gautume efektyvius aukštos kokybės sprendinius. Įvairių optimalaus sprendinio paieškos paradigmų realizavimas bei kelių paradigmų derinimas atskirų klasių uždaviniams spręsti – aktuali praktinė problema, kuriai pastaruoju metu mokslinėje literatūroje skiriama daug dėmesio.

1.3 Tyrimų objektas

Disertacijos tyrimų objektas yra tvarkaraščiai, jų duomenys, įvairių klasių tvarkaraščių struktūros, leistinųjų tvarkaraščių sudarymo algoritmai, optimalių tvarkaraščių paieškos euristiniai algoritmai, jų parametrų derinimo bei kelių algoritmų kombinavimo metodai, paieškos algoritmų sudėtingumo bei efektyvumo analizė.

1.4 Tyrimų tikslas ir uždaviniai

Analizuojant tvarkaraščius, kyla klausimas, kokiu pavidalu pateikti duomenis ir koduoti tvarkaraštį. Sprendžiant optimalaus tvarkaraščio sudarymo uždavinį, reikia rasti tokį vektorių, kurio komponentes sudaro užduočių pradžių laiko momentai, tenkinantys nuoseklumo sąryšius, išteklių ribojimus, o šis vektorius teikia minimalią (maksimalią) pasirinkto kriterijaus reikšmę. Tokiu kriterijumi dažniausiai yra viso projekto įgyvendinimo laikas, kurį reikia minimizuoti. Šis vektorius iš esmės apibūdina tvarkaraštį. Juo remiantis galima rasti užduočių pabaigų laiko momentų vektorių bei apskaičiuoti išteklius, kiekvienu laiko momentu reikalingus vykdant projektą. Tačiau analizinis tvarkaraščio aprašymas, remiantis užduočių pradžių vektoriumi, yra gana nepatogus optimizavimo požiūriu, nes klasikinio matematinio programavimo taikymas šiuo atveju veda į sudėtingą daugelio ekstremumų uždavinį, išsprendžiamą tiktai kombinatorinio perrinkimo būdu. Šioje disertacijoje nagrinėjamas užduočių tvarkaraščio kodavimas užduočių pirmumo sąrašais, bei euristinių paradigmų realizavimas remiantis būtent tokiu tvarkaraščio kodavimu. Sukurtų algoritmų efektyvumui tirti yra naudojamos standartinių testinių uždavinių duomenų bazės. Jos leidžia palyginti įvairių autorių sukurtus algoritmus, sprendžiant tam tikrą seriją uždavinių, artimų realioje praktikoje iškylantiems uždaviniams. Šiame darbe yra nagrinėjama biblioteka PSPLib. Joje galima rasti duomenų rinkinių įvairaus tipo tvarkaraščių su ribotais ištekliais uždaviniams spręsti, o taip pat geriausius žinomus sprendinius. Tyrinėtojai savo algoritmų tikrinimui gali parsisiųsti testų rinkinius bei nusiųsti rezultatus, kurie gali būti patalpinti šioje bibliotekoje.

Šio darbo tikslas – ištirti užduočių tvarkaraščių su ribotais ištekliais sudarymo algoritmus bei sukurti optimalių tvarkaraščių paieškos metodus, besiremiančius euristinėmis paradigmomis, tvarkaraščio kodavimu užduočių pirmumo sąrašu.

Norint pasiekti šį tikslą, reikėjo išspręsti tokius uždavinius:

- Remiantis kodavimu užduočių pradžių vektoriais, išanalizuoti užduočių tvarkaraščio su ribotais ištekliais formuluotę.
- Išnagrinėti ir palyginti įvairių tvarkaraščių su ribotais ištekliais uždavinių klasių struktūras.
- Ištirti tvarkaraščio radimo algoritmus, remiantis matematinio programavimo formuluote ir užduočių kodavimu užduočių pradžių vektoriais.
- Ištirti tinklinio projektavimo metodus (mažėjančių trukmių, kritinio kelio ir kitus).

- Ištirti kombinatorinio perrinkimo metodus bei jų efektyvumą.
- Ištirti optimalių tvarkaraščių paieškos euristines paradigmas.
- Ištirti modeliuojamojo atkaitinimo algoritmo parametrų parinkimo metodą tolydžiojo optimizavimo atveju, pritaikant Pareto tipo tikimybinius skirstinius.
- Pritaikyti sudarytą modeliuojamojo atkaitinimo algoritmą tvarkaraščių diskrečiojo optimizavimo uždaviniams spręsti, remiantis kodavimu užduočių pirmumo vektoriais.
- Sudaryti genetinius algoritmus tvarkaraščiams su ribotais ištekliais rasti, remiantis kodavimu užduočių pirmumo vektoriais.
- Ištirti sudarytų algoritmų efektyvumą, sprendžiant testinius uždavinius iš standartinės duomenų bazės PSPLib.
- Pateikti rekomendacijas euristinės paieškos paradigmoms realizuoti, remiantis užduočių pirmumo vektoriumi.

1.5 Mokslinis naujumas

Remiantis Pareto tipo tikimybiniais skirstiniais bei tvarkaraščio kodavimu užduočių pirmumo sąrašais, pasiūlytas modeliuojamojo atkaitinimo algoritmas tvarkaraščiams optimizuoti. Pasiūlyta genetinio algoritmo realizacija tvarkaraščiams optimizuoti, besiremianti tvarkaraščio kodavimu užduočių pirmumo sąrašais. Eksperimentiškai nustatyta, kaip, priklausomai nuo uždavinio užduočių apimties bei naudojamų išteklių kiekio, parinkti modeliuojamojo arba genetinio algoritmo parametrus. Remiantis užduočių pirmumo vektoriumi, pateiktos rekomendacijos euristinės paieškos paradigmoms realizuoti. Parodyta, jog sudaryti metodai leidžia surasti uždavinių, artimų realiai iškylantiems praktikoje, sprendinius, panaudojant priimtinus kompiuterio laiko ir atminties išteklius.

1.6 Praktinė darbo reikšmė

- Sukurta programinė įranga Delphi, FreePascal, MathCad, C++ priemonėmis, realizuojanti sukurtus metaeuristinius algoritmus, naudojančius darbų pirmumo sąrašą.
- Sudarytas genetinis algoritmas buvo pritaikytas operacijų paskirstymo uždavinio sprendiniui optimizuoti, sprendžiant HP montažinių plokščių surinkimo uždavinį.

1.7 Darbo rezultatų aprobavimas

Tyrimų rezultatai buvo pristatyti ir aptarti šiose nacionalinėse ir tarptautinėse konferencijose Lietuvoje ir užsienyje:

- „Pareto tipo modelių pritaikymas modeliuojamojo atkaitinimo (Simulated Annealing) algoritmuose“. LMD XLIV konferencija. 2003, birželis.
- „Simulated annealing with Pareto models“. Tarptautinė konferencija „21st EURO Summer Institute STOCHASTIC AND HEURISTIC METHODS IN OPTIMIZATION“. 2003, rugpjūtis.
- „Pareto tipo sunkių uodegų modelių taikymas modeliuojamojo atkaitinimo algoritmuose“ (*„Application of Pareto type heavy tailed models in Simulated Annealing algorithms*“). Vokietijos Hamburgo universiteto, Informacinių sistemų instituto mokslinis seminaras. 2004, lapkritis.
- „Tvarkaraščių su ribotais resursais sudarymo algoritmai ir jų taikymas“. KTU konferencija „Informacinės technologijos ‘2005“ (*Iš konferencijų ciklo „Lietuvos mokslas ir pramonė*“). 2005, sausis.
- „Tvarkaraščių su ribotais ištekliais sudarymo algoritmai ir jų tyrimas“. KTU konferencija „Matematika ir matematikos dėstymas – 2005“ sekcijoje „Matematika ir matematinis modeliavimas“. 2005, balandis.
- „Optimality testing in stochastic and heuristic algorithms“. Tarptautinė konferencija EU/ME'2005 (*Workshop of the European Chapter on Metaheuristics „Metaheuristics and Large-Scale Optimization*“), Vilnius, 2005, gegužė.
- „Euristinių metodų taikymas darbų su ištekliais tvarkaraščiams planuoti ir optimizuoti“. KTU konferencija „Matematika ir matematikos dėstymas – 2006“ sekcijoje „Matematika ir matematinis modeliavimas“, 2006, balandis.
- „Tvarkaraščių su ribojimais ištekliams optimizavimas“. Lietuvos jaunųjų mokslininkų konferencija „Operacijų tyrimas ir taikymai“ (LOTD - 2006, Vilnius), 2006, gegužė.

1.8 Darbo rezultatų publikavimas

Tyrimo rezultatai publikuoti šiuose moksliniuose leidiniuose:

Leidiniuose, esančiuose Lietuvos mokslo tarybos patvirtintame tarptautinių duomenų bazių sąraše:

1. G.Felinskas, L.Sakalauskas, „Pareto tipo modeliai modeliuojamojo atkaitinimo algoritmuose“. „Lietuvos Matematikos rinkinys“, 2003, T.43, spec.nr., 573-578. (ISSN 0132-2818) (Duomenų bazėse: CIS, MatSciNet, VINITI, Zentralblatt MATH)
2. L.Sakalauskas, V.Bartkute, G.Felinskas, „Optimality testing in stochastic and heuristic algorithms“. „Technological and economic development of economy“, 2006, XII tomas, Nr.1, 4-10. (ISSN 1392-8619) (Duomenų bazėse: CSA / ASCE Civil Engineering Abstracts, Business source complete (EBSCO), Business Source Premier (EBSCO), ICONDA (IRBdirect), SCOPUS)
3. G.Felinskas, L.Sakalauskas, „Optimization of resource constrained project schedules by simulated annealing and variable neighborhood search“. „Technological and economic development of economy“, 2006, XII tomas, Nr.4., p. 307-313. (ISSN 1392-8619) (Duomenų bazėse: CSA / ASCE Civil Engineering Abstracts, Business source complete (EBSCO), Business Source Premier (EBSCO), ICONDA (IRBdirect), SCOPUS)
4. G.Felinskas, L.Sakalauskas, „Optimization of resource constrained project schedules by genetic algorithm based on the job priority list“. „Information technology and control“, 2006, 35 tomas, Nr.4, 412-418. (ISSN 1392-124X) (Duomenų bazėse: INSPEC, VINITI)

Kituose recenzuojamuose moksliniuose leidiniuose:

- G.Felinskas, L.Sakalauskas, „Tvarkaraščių su ribotais ištekliais sudarymo algoritmai ir jų tyrimas“. „Matematika ir matematinis modeliavimas“, 1, 2005, 105-110. (ISSN 1822-2757)
- L.Sakalauskas, G.Felinskas „Tvarkaraščių su ribotais ištekliais sudarymo euristiniai algoritmai, jų tyrimas bei taikymai“. „Informacijos mokslai“, 2006, 38 tomas, 90-103. (ISSN 1392-1487, 1392-0561) (Duomenų bazėse: CEEOL)

Konferencijų pranešimų medžiagoje:

- G.Felinskas, L.Sakalauskas, „Tvarkaraščių su ribotais resursais sudarymo algoritmai ir jų taikymas“. „Informacinės technologijos ‘2005. Konferencijos pranešimų medžiaga“, Kaunas, Technologija, 2005, 406-414. (ISBN 9955-09-789-2)

1.9 Disertacijos struktūra

Disertaciją sudaro 8 skyriai, literatūros sąrašas ir priedai.

1-asis skyrius yra įvadinis.

2-ajame skyriuje analizuojami tvarkaraščių uždaviniai bei jų sprendimo metodai. Apžvelgiami taikomieji tvarkaraščių uždaviniai įvairiose srityse: transporto, gamybos, užsiėmimų tvarkaraščių, telekomunikacijų ir kitose. Analizuojamos tipinių tvarkaraščių uždavinių formuluotės, aptariamos sąvokos, reikalingos tvarkaraščiams formalizuoti. Pateikiami įvairūs metodai leistiniesiems tvarkaraščiams rasti bei optimizuoti, o taip pat aktualios tvarkaraščių sudarymo problemos, euristinės tvarkaraščių optimizavimo paradigmos. Aptariama ir įvertinama programinė įranga tvarkaraščiams įvairiose srityse sudaryti (mokymo įstaigų užsiėmimų, transporto, projektavimo ir pan.).

3-iajame skyriuje yra nagrinėjami ribotų išteklių tvarkaraščių sudarymo uždavinių (RITSU) formalizavimo klausimai. Pateikiama matematinė RITSU formuluotė, naudojant užduočių trukmių, užduočių pirmumo bei ribojimų sąrašus. Naudojant grafų teorijos elementus, pateikiami formalizuoto tvarkaraščio duomenų pateikimo bei vaizdavimo būdai. Pateikiamos klasikinio ir apibendrintojo RITSU prielaidos. Aptariami galimi tvarkaraščių uždavinių optimizavimo kriterijai, sprendinių vaizdavimas, įvedamas tvarkaraščių formalizavimas užduočių pirmumo sąrašais.

4-ajame skyriuje yra nagrinėjami tvarkaraščių uždavinių perrinkimo ir leistinųjų sprendinių radimo metodai. Aptariami sveikaskaičiai optimizavimo uždaviniai susiję su tvarkraščių planavimu (keliaujančio pirklio uždavinys, kuprinės uždavinys, nepriklausomų užduočių tvarkaraščių uždaviniai ir kiti), perrinkimo uždaviniai ir jų sudėtingumas. Taip pat pateikiami perrinkimo algoritmų kompiuterinio modeliavimo rezultatai, nustatant ribines sveikaskaičių ir tvarkaraščių uždavinių (realizuojamų kompiuteriu panaudojant priimtinus kompiuterio atminties bei laiko išteklius) dimensijas. Išnagrinėti tvarkaraščių uždavinių leistinųjų sprendinių radimo metodai (mažėjančių trukmių algoritmas, kritinių kelių metodas), skirti pradiniams artiniams euristinės paieškos algoritmuose gauti.

5-ajame skyriuje aptariamos euristinės paradigmos, taikomos sveikaskaičio programavimo bei tvarkaraščių sudarymo uždaviniams spręsti. Yra aprašyti godieji algoritmai ir jų taikymai, pateikiamos modeliuojamojo atkaitinimo, paieškos su draudimais, genetinių algoritmų, išbarstytosios paieškos bei kintamos aplinkos paieškos metodo paradigmos.

6-ajame skyriuje yra nagrinėjami stochastiniai ir euristiniai tvarkaraščių optimizavimo metodai, naudojantys užduočių pirmumo sąrašą. Pateikiami nuoseklaus sąrašo dekodavimo ir sąrašo leistinumo nustatymo algoritmai. Formuluojamas tvarkaraščių atsitiktinės paieškos Monte-Karlo metodu algoritmas. Sudaromas tvarkaraščio optimizavimo modeliuojamojo atkaitinimo ir kintamos aplinkos algoritmas. Sudaromas tvarkaraščių optimizavimo genetinis algoritmas, besiremiantis pirmumo sąrašu.

7-ajame skyriuje yra nagrinėjama tvarkaraščių optimizavimo euristiniais metodais eksperimentinio tyrimo metodologija ir pateikiami euristinių tvarkaraščių optimizavimo algoritmų testavimo ir praktinio taikymo rezultatai. Pateikiamas PSPLib bibliotekos aprašymas, parametrų nustatymai, informacija apie uždavinių duomenų ir duomenų apie sprendinius gavimą, testinių uždavinių charakteristikos ir pan. Yra pateikiami sukurtųjų modeliuojamojo atkaitinimo ir genetinės paieškos algoritmų testavimo rezultatai, sprendžiant bibliotekos PSPLib testinius uždavinius. Remiantis testavimo rezultatais yra pateikiamos rekomendacijos praktiniam euristinių algoritmų realizavimui bei taikymui. Pateikiami HP montažinių plokščių surinkimo uždavinio sprendimo rezultatai.

8-ajame skyriuje pateikiamas gautų rezultatų apibendrinimas, išvados bei rekomendacijos.

Disertacijos pabaigoje pateikiamas literatūros sąrašas ir priedai.

Prieduose pateikiami grafų teorijos pagrindiniai elementai ir sąvokos, informacija apie metrines ir topologines erdves bei algoritmų sudėtingumą, PSPLib pradinių duomenų rinkinių aprašymai, tvarkaraščių optimizavimo, naudojantis pirmumo sąrašu, programinės įrangos aprašymai.

Padėka

Labiausiai dėkoju savo moksliniam vadovui prof. habil. dr. Leonidui Sakalauskui. Nuoširdus AČIŪ už atvedimą į mokslinių ieškojimų kelią, už visokeriopą pagalbą, už patarimus ir pasiūlymus, už reiklumą ir griežtumą tada, kai pristigdavo jėgų ar motyvacijos, už padrąsinimą ir pasitikėjimą manimi, už nuolatinį skatinimą tobulėti ir plėsti savo akiratį ne tik dalykinėje srityje.

Taip pat dėkoju prof. habil. dr. J. Mockui, dr. J.Žilinskui už vertingą kritiką, pastabas ir diskusijas. Dėkoju MII duomenų analizės skyriaus operacijų tyrimo sektoriaus darbuotojams bei doktorantams už bendradarbiavimą, pagalbą ir supratimą. Taip pat visiems, kurie prisidėjo prie šio darbo vertingomis pastabomis ir pasiūlymais.

Dėkoju Šiaulių universiteto informatikos katedros vedėjai Sigitai Turskienei už nuolatinį palaikymą ir domėjimąsi darbų sėkme, už supratimą ir skatinimą. Taip pat dėkoju visiems katedros kolegoms už moralinį palaikymą, pagalbą, už prablaškymą nemokslinėje veikloje.

Noriu padėkoti Lietuvos valstybiniam mokslo ir studijų fondui už suteiktą finansinę paramą disertacijos rengimo metu. Dėkoju savo šeimai už ketverių metų kantrybę, supratingumą ir skatinimą, ačiū tėvams ir draugams už palaikymą.

2 skyrius. Tvarkaraščių uždavinių analitinis tyrimas

2.1 Įvadas

Tvarkaraščių uždavinių klasifikacija buvo nagrinėjama įvairiais požiūriais daugelyje straipsnių ir disertacijų (Lageweg, Lawler, Lenstra, Rinnooy Kan, 1982, Reklaitis, 1982, Herrmann, Lee, Snowdon, 1993, Liu, Maccarthy, 1996, Kolisch, Hartmann, 1999, Blazewic, Ecker, Pesch, Schmidt, Weglarz, 2001, Yang, Geunes, O'Brien, 2001, Yamada, 2003, Pinedo, 2005).

Klasifikuojant tvarkaraščių uždavinius, juos galima skirstyti praktinių problemų, kuriuose jie kyla, požiūriu, arba tvarkaraščių uždavinių formalizavimo bei sprendimo metodologijos požiūriu. Dabar apžvelgsime tvarkaraščių skirstymą taikomųjų uždavinių požiūriu, po to panagrinėsime tvarkaraščių tipinių uždavinių klasifikaciją. Tvarkaraščių uždavinių sprendimo metodologiją apžvelgsime 2.4 skyrelyje.

2.2 Tvarkaraščių taikomieji uždaviniai

Tvarkaraščių radimo uždaviniai iškyla įvairiose veiklos srityse: gamyboje, ryšių ir kompiuterių tinklų architektūrai parinkti bei informacijos srautams valdyti, logistikoje, vadyboje, versle, projektavime, ekonomikoje, ir kitose srityse.

Gamyboje galime susidurti su tokiais uždaviniais, kai įmonei reikia optimizuoti gamybos planą, kiek kokios produkcijos vienetų pagaminti, maksimizuojant pelną. Pavyzdžiui, įmonė gamina tam tikrą produkciją, kurios gamybai naudojamos tam tikros medžiagos (žaliavos), žinomi jų medžiagų poreikiai bei turimi ištekliai, taip pat žinomas pelnas už produkciją. Pelną reikia maksimizuoti atsižvelgiant į tai, jog medžiagų sąnaudos neviršytų jų išteklių. Jei prie visų šių sąlygų dar prijungti transporto priemonių paskirstymą produkcijos išvežiojimui į sandėlius ar į prekybos vietas, jų maršrutizavimą, turėsime pakankamai sudėtingą darbų tvarkaraščių planavimo uždavinį gamyboje.

Gamybos procesų organizavimo uždaviniuose informacijos srautai apima duomenis apie užduotis (darbus), jų trukmes, nuoseklumo sąryšius, būtinus laukimo laikus, o taip pat apie vykdytojus, įrenginius, procesorius ir kitus pradinius duomenis ir ribojimus. Tokių uždavinių tikslas yra nustatyti tokią darbų atlikimo eigą vykdytojams, kad būtų nenusižengiama įvestiems ribojimams ir stengiamasi optimizuoti projekto atlikimo laiką, sąnaudas, kaštus ir pan. Šiam tikslui siekti būtina turėti algoritmus, kurie būtų tinkami taikyti sudėtingose planavimo ir darbų kalendorinio grafiko sudarymo sistemose. Su gamybos procesų organizavimu glaudžiai susiję yra inžinerinio projektavimo uždaviniai, kurie apima projekto vykdymo grafikų sudarymą (Badri, Davis, Davis, Hollingsworth, 1998, Fink, Voss, 2003, Jozefowska, Mika, Rozycki, Waligora, Weglarz, 1998, Kim, Moon, 2003). Ypač dažnai tokie uždaviniai yra sprendžiami statybų planavimo metu (Kalanta, 2003, Pušinaitis, 2004). Tinklinio planavimo uždaviniai taip pat buvo

nagrinėjami disertacijose ir moksliniuose darbuose (Vaičiulis, Matulis, 1970, Baskas, 1973, Vaičiulis, 1973).

Telekomunikacijų tinkluose dažnai sprendžiami įvairių informacinių srautų apdorojimo uždaviniai. Tokie uždaviniai ypač svarbūs GSM tinkluose, kai tenka apdoroti įvairius duomenų paketus, analizuoti pliūpsninius srautus, kontroliuoti pranešimų srautus, kad jie patikimai patektų pas gavėjus su didžiausia sparta ir mažiausiu vėlavimu. Su panašiais uždaviniais susiduriama ir kompiuterių tinkluose, kai reikia apdoroti siunčiamų duomenų srautus. Kuriant ir tobulinant daugiaprocesorines sistemas taip pat yra svarbūs įvairūs informacinių paketų organizavimo ir skirstymo uždaviniai. Projektuojant ir valdant kompiuterių tinklus, tenka spręsti informacijos perdavimo patikimumo ir spartos problemas (Azarmi, Smith, 1995, Nemeth, Lovrek, Sinkovic, 1997, Shade, Orman, 1997, Lesaint, Voudouris, Azarmi, 2000).

Transporto organizavimo, logistikos, koordinavimo uždaviniai taip pat yra vieni iš aktualiausių. Šio tipo uždaviniams priklauso transporto priemonių parinkimo ir paskirstymo uždaviniai, kai iš įvairių galimų krovinio vežimo priemonių bei maršrutų reikia parinkti minimizuojančius vežimo išlaidas ar laiką. Dažnai transporto uždaviniai yra sprendžiami kartu su gamybos organizavimo uždaviniais, įtraukiant ir tiekimo uždavinius. Pastarieji kombinuotieji uždaviniai yra vadinami tiekimo grandinių optimizavimu (Supply Chain Management) (Becker, Smith, 1997, Handfield, Nochols, 1999, Shapiro, 2001, Supply Chain Management Research Center 2006).

Viešajam transportui mieste organizuoti reikia spręsti pakankamai sudėtingus kombinuotus uždavinius, kuriuose reikia optimizuoti transporto priemonių maršrutus, minimizuojant sąnaudas, maksimizuojant teritorijos padengiamumą, optimizuojant transporto priemonių kiekį keleivių srautams, sudarant tinkamas laiko lenteles, tuo minimizuojant laukimo laiką (Voss, 1992, Ceder, Golany, Tal, 2001, Fores, Proll, Wren, 2002).

Su panašiais, tik su sudėtingesniais uždaviniais susiduriama traukinių eismo reguliavimo, lėktuvų skrydžių valdymo uždaviniuose, kur ypač svarbu tikslumas ir patikimumas (Laplagne, Kwan, 2005).

Laivų uostuose dažnai susiduriama su konteinerių krovos, sandėliavimo, jų tvarkymo, išvežimo uždaviniais. Labai svarbu tinkamai organizuoti šiuos krovos darbus, kad kroviniai kuo trumpiau užsibūtų uostuose. Kai kuriuose pasaulio uostuose šie darbai beveik visiškai atomatizuoti ir valdomi dinaminių kompiuterinių sistemų, kurios organizuoja konteinerių rikiavimą, atsižvelgiant į jų laukiančias transporto priemones ir pan. (Kim, Moon, 2003, Steenken, Voss, Stahlbock, 2004, Günther, Kim, 2006).

Be tvarkaraščių sudarymo neišsiversime ir **mokymo įstaigose** (Yoshikawa, Kaneko, Yamanouchi, Watanabe, 1996, Schroth, 1997, Smykalov, 2001, Solodovnikova, Solodovnikov, 2002). Ypač sudėtingi yra aukštųjų ir profilinių mokyklų užsiėmimų tvarkaraščių sudarymo uždaviniai. Dažniausiai tokiuose uždaviniuose reikia suplanuoti ir optimizuoti savaitinius

užsiėmimų tvarkaraščius studentų ar moksleivių grupėms, priskiriant grupei ar pogrupiui dalyko dėstytoją ar mokytoją, patalpas, tenka atsižvelgti į įvairius ribojimus, minimizuoti „langų“ skaičių per dieną studentams ar dėstytojams.

Universiteto užsiėmimų tvarkaraščių sudarymo uždavinys yra bene labiausiai tyrinėjamas iš visų uždavinių, susijusių su akademiniu gyvenimu (Bardadym, 1996, Burke, Jackson, Kingston, Weare, 1997, Carter, Laporte, 1998, Petrovic, Burke, 2004). Bardadym suklasifikavo universitetų tvarkaraščių sudarymo uždavinius į 5 grupes (Bardadym, 1996):

- Fakultetų tvarkaraščiai: reikia paskirti tam tikros kvalifikacijos dėstytojus numatomiems kursams.
- Klasių-mokytojų tvarkaraščiai: paskirstyti užsiėmimus, kai mažiausias vienetas tvarkaraštyje yra studentų ar moksleivių klasė (grupė).
- Užsiėmimų tvarkaraščiai: paskirstyti užsiėmimus, kai mažiausias vienetas tvarkaraštyje yra atskiras studentas ar moksleivis.
- Egzaminų (sesijų) tvarkaraščiai: paskirstyti egzaminus studentams taip, kad studentai neturėtų dviejų egzaminų tuo pačiu metu.
- Kabinetų skirstymas: susiejus klases ir mokytojus, šias klasė-mokytojas poras paskirti kabinetams.

2.3 Tvarkaraščių tipiniai uždaviniai

Tvarkaraščių uždaviniai, kurie iškyla įvairiose srityse, dažnai turi bendrų ypatybių ir atsiribojant nuo antraeilių kiekvienos srities detalių galima formuluoti tipinius tvarkaraščių uždavinius. Uždavinių formalizavimas kiekvienoje konkrečioje srityje padeda priskirti juos tipinių uždavinių grupei ir parinkti tinkamus sprendimo metodus. Dažnai būna, jog iš pažiūros skirtingų taikomųjų sričių uždaviniai gali būti suvedami į tą pačią uždavinių klasę. Tvarkaraščių uždaviniai pasižymi didele įvairove, todėl, priklausomai nuo informacijos, reikalingos tvarkaraščiui apibūdinti, juos galima įvairiai klasifikuoti. Informacija, reikalinga tvarkaraščiams sudaryti, apima duomenis apie užduočių trukmes, jų nuoseklumo sąryšius, naudojamus išteklius užduotims atlikti, išteklių atsargas bei ribojimus, procesorių, atliekančių užduotis, pajėgumą, tvarkaraščio realizavimo ypatybes ir panašiai.

Dažniausiai išskiriami tokie kriterijai tvarkaraščiams klasifikuoti:

- Statinis arba dinaminis duomenų srauto pobūdis,
- Išteklių tipai,
- Ribojimų pobūdis,
- Duomenų apibrėžtumas.

2.3.1 Statiniai tvarkaraščiai

Tvarkaraščių planavimas gali būti statinis arba dinaminis. Statinio planavimo atveju naudojant fiksuotus ir iš anksto žinomus duomenis apie darbų srautą, reikia parinkti darbų eiliškumą, atsižvelgiant į ribojimus, susijusius su darbų nuoseklumo sąryšiais, medžiagų, žaliavų ar finansų ištekliais, vykdytojų gebėjimus atlikti darbus ir pan. Tokie uždaviniai sprendžiami tinklinio arba kalendorinio planavimo metodais. Su šiais uždaviniais dažniausiai susiduriama organizuojant įvairius statybų arba gamybos organizavimo darbus (Kalanta, 2003, Pušinaitis, 2004). Sudarant kalendorinį grafiką tokiems darbams yra siekiama keleto tikslų:

- nustatyti kiekvieno statybos arba gamybos proceso darbų atlikimo trukmę;
- technologiniu eiliškumu suderinti atskirų darbų atlikimą;
- racionaliai paskirstyti darbo jėgos ir kitus resursus;
- nustatyti optimalią bendrą darbų atlikimo trukmę.

Galima išskirti keletą kalendorinių grafikų rūšių:

- statybos arba gamybos proceso kalendorinis grafikas;
- ciklo darbų kalendorinis grafikas;
- viso projekto vykdymo kalendorinis grafikas;
- suvestinis kalendorinis grafikas.

Proceso kalendorinis grafikas sudaromas atskiram statybos arba gamybos procesui vykdyti, sudarant technologines korteles, kurios yra sudedamoji darbų technologijos (vykdymo) projekto dalis. Ciklo darbų kalendorinis grafikas sudaromas atskiro etapo darbams vykdyti. Atskiro proceso ir ciklo darbų kalendoriniai grafikai skiriasi tuo, kad proceso kalendorinis grafikas sudaromas tik vienam procesui, o ciklo darbų kalendorinis grafikas sudaromas keletui technologiniu eiliškumu susijusiems statybos arba gamybos procesams. Suvestiniai kalendoriniai grafikai sudaromi visų ciklų darbams vykdyti.

Pavyzdžiui, statyboje suvestinį kalendorinį grafiką gali sudaryti (Pušinaitis, 2004):

- ciklo darbų kalendoriniai grafikai (pavyzdžiui, kuomet statybos bendrovė specializuojasi atskirų pastato dalių statyba);
- pastato statybos kalendoriniai grafikai (kuomet statybos bendrovė atlieka visus pastatų bendruosius statybos darbus);
- mišrūs kalendoriniai grafikai (kuomet statybos bendrovė atlieka atskirų pastato dalių ir visus pastatų bendruosius statybos darbus).

Išeities duomenys kalendoriniams grafikams sudaryti statyboje yra šie: duomenys apie galimybę aprūpinti pastato statybą materialiniais, techniniais ir darbo jėgos resursais; duomenys apie statybinės organizacijos sudarytus kontraktus su užsakovais; pastato darbo projektas; pagrindiniams statybos procesams sudarytos technologinės kortelės.

Kalendoriniai grafikai yra sudaromi tinklinio planavimo metodais, atvaizduojant atliekamus darbus kryptiniu grafu. Tinklinių grafikų vaizdavimo būdus aptarsime 3.3 skyrelyje.

2.3.2 Dinaminiai tvarkaraščiai

Dinaminiuose uždaviniuose tenka priimti sprendimus atsižvelgiant į turimus darbo jėgos, techninių įrengimų, laiko bei finansinius ribojimus, apdorojant duomenų srautą apie iškylančius darbus. Tokio tipo uždaviniai dažnai iškyla transporto, krovos, gamybos organizavimo metu, telekomunikacijų ir ryšių tinklų valdymo metu ir pan. Dinaminiuose uždaviniuose informacija apie darbus gali būti iš anksto nežinoma ir gali paaiškėti tik proceso vykdymo metu.

Su dinaminiais uždaviniais yra artimai susiję realaus laiko uždaviniai, kai sprendimą apie darbų organizavimą reikia priimti operatyviai, nes laikas sprendimams priimti yra ribojamas. Tokio pobūdžio darbai iškyla organizuojant darbų srautų (konvejerių) darbą (*flowshop*) (Frostig, Adiri, 1985, Tandon, Cummings, La Van, 1991, Shmoys, Stein, Wein, 1994, Hall, 1995, Aldowaisan, Allahverdi, 2003). Pastarieji uždaviniai pasižymi tuo, jog juose nustatytas tam tikras procesorių (mašinų, vykdytojų) skaičius. Visi darbai atliekami ta pačia tvarka, nuosekliai apdorojant kiekvieną darbą kiekvienu procesoriumi.

Atskiru atveju yra nagrinėjami lankstieji (*flexible flowshop*) (Wittrock, 1988, Wang, Li, 2002) srautinių darbų uždaviniai su nuosekliomis pakopomis, kurių kiekvienoje yra tam tikras skaičius lygiagrečių procesorių. Kiekvienoje pakopoje darbui atlikti reikia tik vieno procesoriaus. Planuojant srautinius darbus taip pat pasinaudojama tuo, jog dalis darbų gali neturėti nuoseklumo saryšių (*OpenShop*) (Sevastjanov, Woeginger, 1998) arba gali būti grąžinti pakartotinai aptarnauti į ta patį procesorių (*Jobshop*) (Brucker, Schlie, 1990, Vaessens, 1995, Sviridenko, Jansen, Solis-Oba, 1999). Šiuo atveju tvarkaraštis nusako maršrutą kiekvienam darbui, t.y. nurodo kuria tvarka jis bus apdorojamas procesoriais.

Kadangi srautiniuose uždaviniuose atsiranda darbų eilės, planavimo metu reikia atsižvelgti į darbų vykdymo eilės tvarką. Dažniausiai eilės yra tvarkomos pagal taisyklę *pirma atėjo – pirma tvarkoma* (FIFO – First In First Out).

2.3.3 Tvarkaraščiai su įvairiais išteklių tipais

Tvarkaraščių uždavinius mes galime susiskirstyti pagal tai, su kokiais ištekliais operuojama uždavinyje. Išskirkime šias grupes:

- **Uždaviniai su ribojimais vykdymui** – apima uždavinius su darbų pertraukimais ir darbo jėgos arba įrengimų ribojimais. Tokiuose uždaviniuose yra tiriamas vykdymo resursų skirstymas statinių ir dinaminių uždavinių atvejais.
- **Uždaviniai su medžiagų ribojimais** – išskiriami uždaviniai tiek su medžiagų stygiumi, tiek su jų pertekliumi, taip pat su medžiagų kokybės tikrinimu. Su šiais uždaviniais dažniausiai susiduriame, kai reikia spręsti racionalaus medžiagų atsargų skirstymo uždavinius.

- **Uždaviniai su ribotais žinių ištekliais** – apima uždavinius, kuriuose tenka priimti sprendimus, kai nepakanka žinių turimai informacijai apdoroti arba žinios yra pasenusios ir nebetinka naujai situacijai. Žinių ištekliais gali būti faktai, technologijos, strategijos. Svarbu žinių išteklius vystyti ir tobulinti, nes tai garantuotų įvairių tvarkaraščių uždavinių sprendinio suradimą.

Uždaviniai su išteklių ribojimais dažnai yra sprendžiami pasirinkus ekonominius kriterijus ir atsižvelgiant į finansinę ribojimų išraišką.

2.3.4 Tvarkaraščių ribojimų tipai

Ribojimai tvarkaraščiuose gali būti labai įvairūs. Gali būti, jog tam tikru laiko momentu nepakaks kokių nors išteklių ar bus nusižengta nuoseklumo sąryšiams, gali būti reikalaujama paisyti tam tikrų laiko terminų ir panašiai (Duncan, 1990, Artiouchine, Baptiste, 2005).

Pagal ribojimų sritis (sferas), tvarkaraščių uždavinius suskirstysime į šias grupes:

- **Tvarkaraščiai su laiko ribojimais** – apima uždavinius su laiko reikalavimais darbų pradžiai ar pabaigai. Su vienokiais ar kitokiais laiko ribojimais susiduriama visuose tvarkaraščių uždaviniuose.
- **Tvarkaraščiai su išteklių ribojimais** – apima nekintamus, ilgalaikius ar fizinius ribojimus medžiagoms, darbo jėgai bei techniniams įrengimams.
- **Tvarkaraščiai su darbų nuoseklumo ribojimais** – apima uždavinius su darbų eiliškumo ribojimais. Atskiru atveju gali būti nagrinėjami nepriklausomų užduočių tvarkaraščiai, kuriuose nėra ribojimų užduočių eiliškumui. Tokie uždaviniai gali būti suvedami į kuprinės arba pakavimo uždavinius (žiūr. 4.1 skyrelį).

Dažniausiai yra taip, jog kuo daugiau yra nuoseklumo sąryšių, tuo sunkiau sudaryti gerą tvarkaraštį.

Pagal ribojimų ypatybes tvarkaraščių uždavinius galime suskirstyti į šias grupes:

- **Tvarkaraščiai su laikinais ribojimais** – ribojimai tokiuose uždaviniuose yra nusakyti tik tam tikriems laiko periodams ir suvaržo procesų pradžios, pabaigos laiko momentus, trukmes, taip pat ir resursų paskyrimus. Kai užduotis tokiuose uždaviniuose jau apdorojama, laikini ribojimai laikomi patenkintais ir naikinami arba atnaujinami.
- **Tvarkaraščiai su nenumatytais ribojimais** – ribojimai tokiuose uždaviniuose nusako netikėtus ar iš anksto nenumatytus ribojimus, kurie keičia įvairius prioritetus ir išteklių skirstymą. Iškilus naujiems ribojimams, atsižvelgiama į turimus ir einama prie tvarkaraščio perdarymo (rescheduling).
- **Tvarkaraščiai su nuolatiniais (ilgalaikiais) ribojimais** – dažniausiai tai fiziniai ribojimai, įtakojantys tvarkaraščio sudarymo procesą, resursų skirstymą. Įprastai šių ribojimų neįtakoja kiti ribojimai.

Ribotų išteklių tvarkaraščių sudarymo uždaviniai yra vieni iš svarbiausių tvarkaraščių teorijos uždavinių, kurie plačiai taikomi praktikoje (Kolisch, 1996a, Hartmann, 1998, Brucker, Knust, Schoo, Thiele, 1998, Bouleimen, Lecocq, 1998, Baar, Brucker, Knust, 1999, Klein, 2000, Hartmann, Kolisch, 2000, Eisenberg, 2002, Kolisch, Hartmann, 2005). Mokslinėje literatūroje pažymima, jog sudarant tokių uždavinių efektyvius sprendimo algoritmus lieka dar daug neišspręstų klausimų. Šių uždavinių formuluotę bei sprendimo metodologiją apžvelgsime 3 skyriuje.

2.3.5 Tvarkaraščių duomenų apibrėžtis

Tvarkaraščių uždaviniai gali būti klasifikuojami pagal duomenų apibrėžties pobūdį:

- **Deterministiniai uždaviniai** – šiuose uždaviniuose yra žinoma visa informacija, reikalinga tvarkaraščiui sudaryti (Gonzalez, 1977, Coffman, 1982, Lawler, Sahni, 1990, Parker, 1995).
- **Stochastiniai uždaviniai** – šiuose uždaviniuose duomenys yra aprašomi tikimybiniais modeliais (Glazebrook, 1979, Schopf, Berman, 1999, Möhring, Schulz, Uetz, 1999, Sakalauskas, 2002).
- **Nedeterministiniai (nestochastiniai) uždaviniai** – šiuose uždaviniuose dalis duomenų yra neapibrėžti ir neapibrėžtis yra aprašoma netikimybiniais modeliais (blausioji logika, intervalinė aritmetika ir pan.) (De Gloria, Faraboschi, Olivieri, 1992, Bilkay, Anlagan, Kilic, 2004).
- **Nepilnai apibrėžti uždaviniai** – šiuose uždaviniuose dalis duomenų gali būti nežinoma, praleista arba turėti klaidų (Ahuja, Clark, Rogers, 1995, Bast, 1998, Shrivastava, Earlie, Dutt, Nicolau, 2004).

Šiuo metu yra labiausiai ištirti įvairių tipų deterministiniai tvarkaraščių sudarymo uždaviniai. Apžvalgose yra pažymima, jog tiriant stochastinių ir nedeterministinių (nestochastinių) tvarkaraščių sudarymo metodus lieka dar daug neišspręstų uždavinių. Nepilnai apibrėžti uždaviniai yra sprendžiami suvedant juos į jau minėtus deterministinius arba nedeterministinius uždavinius.

2.4 Tvarkaraščių uždavinių sprendimo metodologija

Tvarkaraščiams sudaryti yra naudojami įvairūs algoritmai, apdorojantys formalizuotus tvarkaraščio pradinius duomenis. Didžioji dauguma tvarkaraščių uždavinių sprendimo algoritmų priklauso NP sudėtingumo klasei. Todėl tikslius tokių uždavinių sprendinius per priimtiną skaičiavimų laiką galime rasti tik mažos apimties uždaviniams. Praktinėse situacijose dažniausiai tenka tenkintis apytikriu sprendiniu, siekiant, kad jis būtų kuo artimesnis optimaliam. Priimtinus praktiniu požiūriu sprendinius galima gauti naudojant daugelį klasikinių *sveikaskaičio programavimo algoritmų*. Sprendinį, gautą klasikiniu algoritmu, galima pagerinti įvairiais *euristiniais algoritmais*, kurie yra kuriami imituojant biologinių sistemų elgseną, naudojant dirbtinio intelekto (DI) principus. Testuojant sukurtus algoritmus bei tiriant jų efektyvumą yra

pasinaudojama standartinių testinių pavyzdžių duomenų bazėmis, ir apie sukurtų algoritmų privalumus yra sprendžiama lyginant didelio skaičiaus testinių uždavinių sprendimo rezultatus. Yra siekiama, kad sukurti algoritmai pagerintų geriausius žinomus testinių uždavinių sprendinius arba pasiektų bent tą patį rezultatą su mažesnėmis skaičiuojamosiomis sąnaudomis. Standartinės testinės duomenų bazės yra aptariamos 7 skyriuje.

2.4.1 Sveikaskaičio programavimo metodai

Diskretieji optimizavimo uždaviniai yra tokie, kurių sprendiniai yra tam tikros diskrečios aibės reikšmės. Kai ši aibė yra sveikieji skaičiai, tada uždaviniai vadinami sveikaskaičiais programavimo uždaviniais. Kita vertus, kombinatoriniai optimizavimo uždaviniai yra tokie, kuriuose reikia parinkti geriausią kombinaciją iš visų galimų. Daugelis kombinatorinių uždavinių gali būti formuluojami kaip sveikaskaičio programavimo uždaviniai. Pagrindinė tokių uždavinių problema yra ta, jog nėra jokių duoto leistino sprendinio optimalumo patikrinimo sąlygų. Tam, kad garantuoti leistinojo sprendinio optimalumą, reikia jį palyginti su visais leistinaisiais sprendiniais, t.y. perrinkti visas galimas alternatyvas, o tai atlikti pernelyg sudėtinga skaičiavimo laiko prasme (NP-pilnas sudėtingumas). Todėl reikia palyginti bent dalį galimų sprendinių.

Praktikoje galima sutikti labai įvairių sveikaskaičio optimizavimo uždavinių. Sveikaskaičių arba diskrečių optimizavimo uždavinių įvairovė yra gana didelė. Tarp jų rasime žemės ūkio gamybos planavimo uždavinius, gatvių eismo reguliavimo signalų sinchronizacijos uždavinius, telekomunikacijų ir kompiuterių tinklų projektavimo uždavinius ir valdymo uždavinius, mokymo įstaigų (pagrindinių ir profilinių mokyklų, kolegijų, universitetų) užsiėmimų tvarkaraščių sudarymo ir optimizavimo uždavinius, įvairūs konteinerių krovos ir jų talpinimo uždaviniai ir dar daug kitų. Aptarsime įvairius uždavinius šiek tiek detaliau. Kai kuriuos uždavinius sveikaskaičiais galima laikyti netiesiogiai (pvz., transporto uždavinys, kai ištekliai ir poreikiai yra sveikieji skaičiai bei kiti tokio tipo uždaviniai), nes jų matematiniuose modeliuose nėra reikalavimo, jog kintamieji būtų sveikieji skaičiai. Kitas pavyzdys – sukirpimo uždaviniai (siuvyklos uždavinys). Juos irgi reiktų priskirti prie sveikaskaičių, nes minimizuojant atliekas siekiam gauti reikiamą sveikąjį skaičių pusfabrikačių. Išskirkime keletą grupių kitų, tiesiogine prasme sveikaskaičių, optimizavimo uždavinių:

1. Uždaviniai su nedalomais objektais. Tai modeliai, aprašantys nedalomos (vienetinės produkcijos) gamybos planus ar planus gamybos, kurioje naudojami nedalomi objektai. Jais gali būti staklės, įrengimai, transporto priemonės. Todėl įvairūs vežimų uždaviniai formuluojami kaip sveikaskaičiai optimizavimo uždaviniai su nedalomais objektais. Pavyzdžiui, krovinių vežimo priemonių optimizavimo, nustatytam vežimų tvarkaraščiui vykdyti reikalingų transporto priemonių (traukinių, sunkvežimių, laivų, lėktuvų) minimalaus kiekio nustatymo uždaviniai, taip pat trumpiausio automobilių kelio, nuvažiuoto vykdant nustatytą krovinių vežimo planą, nustatymo uždaviniai.

2. Kombinatoriniai uždaviniai. Prie jų priskiriamas trumpiausio maršruto tarp dviejų punktų nustatymo uždavinys, plačiai žinomas keliaujančio pirklio uždavinys, daugelis kalendorinio planavimo uždavinių ir kiti. Keliaujančio pirklio modelis aprašo krovinio pristatymo visiems vartotojams trumpiausio maršruto nustatymo uždavinį (pavyzdžiui, prekių tiekimas iš sandėlio visam parduotuvių tinklui, o taip pat ir šiukšlių surinkimas ir išvežimas bei kiti uždaviniai).
3. Gamybos objektų bei gamybos priemonių optimalaus išdėstymo uždaviniai, įvairūs specializacijos, kooperacijos ir kiti uždaviniai.

Sveikaskaičio programavimo taikymo tvarkaraščių sudarymo uždaviniams spręsti klausimai yra detaliau apžvelgiami 3 ir 5 skyriuje.

2.4.2 Euristinės tvarkaraščių optimizavimo paradigmos

Sprendžiant optimizavimo uždavinius tenka naudoti įvairius euristinius paieškos metodus. Vienos iš žinomiausių tokių metodų paradigmų yra šios:

- Modeliuojamojo atkaitinimo (Simulated Annealing) algoritmas (Kirkpatrick, Gelatt Jr., Vecchi, 1983, Dorea, 1997, Locatelly, 2000a),
- Genetiniai algoritmai (genetic algorithms) (Goldberg, 1989, Khouja, Michalewicz, Wilmot, 1998, Glibovec, Medvidj, 2003),
- Paieška su draudimais (Tabu Search) (Glover, 1990, Bataiti, Tecchiolli, 1994, Glover, Laguna, 1997, Chelouah, Siarry, 2000, Machado, Shiyou, Ho, Peihong, 2001),
- Skruzdžių kolonijos (Ant colony systems) (Colorni, Dorigo, Maniezzo, 1991, Dorigo, Gambardella, 1997, Chu, Roddick, Pan, 2004),
- Memetiniai algoritmai (Memetic algorithms) (Moscato, 1989, Radcliffe, Surry, 1994),
- Išbarstytoji, išsklaidytoji paieška (Scatter search) (Glover, 1998, Glover, Laguna, Martí, 2003),
- Spiečių algoritmai (Swarm algorithms) (Eberhart, Kennedy, 1995, Clerc, 2004, Zhang, 2005),
- Dirbtinių neuroninių tinklų metodai (Neural networks) (Gulati, Iyengar, Toomarian, Protopopescu, Barhen, 1987, Hemani, Postula, 1990, Cavalieri, Mirabella, 1996, Luh, Zhao, Thakur, Chen, Chieu, Chang, 1999).

Euristinio optimizavimo paradigmos detaliau aptariamos 5 skyriuje.

2.5 Tvarkaraščių sudarymo programinė įranga

Įvairių tipų tvarkaraščiams sudaryti yra naudojama speciali programinė įranga, kuri skirta mokymo įstaigų užsiėmimų, transporto srautų, projektavimo darbų, o taip pat bendro pobūdžio

tvarkaraščiams sudaryti. Galima būtų išskirti tokias programinės įrangos, skirtos įvairiems tvrakaraščių uždaviniams spręsti, grupes:

- Užsiėmimų tvarkaraščių programinė įranga.
- Transporto tvarkaraščių sudarymo programinė įranga.
- Projektavimo ir gamybos procesų tvarkaraščių sudarymo programinė įranga.
- Sveikaskaičio modeliavimo ir optimizavimo programinės sistemos.
- Ir kitos.

Įvairių mokymo įstaigų užsiėmimų tvarkaraščių sudarymas yra gana sudėtinga problema. Nors dar daug kur tvarkaraščiai yra sudaromi rankiniu būdu, programinės šių uždavinių sprendimo sistemos taikomos vis plačiau. Daugelis šių sistemų yra skirtos vaizdžiai pateikti sudarytus tvarkaraščius ir reikalauja daug rankinio darbo. Viena iš plačiai naudojamų sistemų yra mokyklų pamokų tvarkaraščių sudarymo sistema „Mimosa“, kuri turi plačias konfigūravimo galimybes ir patogią vartotojo sąsają (Mimosa Scheduling Software, 2007). „MIMOSA“ sudėlioja tvarkaraštį pagal nurodymus, įvestus rankiniu būdu. Tačiau sukurtas tvarkaraštis gali turėti daug „langų“ ir kitokių nepatogumų. Programoje yra patogi grafinė aplinka tam, kad perdėlioti ir derinti tvarkaraštį. Tačiau suvesti pradinius duomenis nėra lengva. Ši programa reikalauja daugiau nei 50% žmogiškųjų išteklių. Programoje nenaudojami jokie sudėtingi optimizavimo metodai. Labiausiai šis įrankis tinka tais atvejais, kai reikia tvarkaraštį koreguoti rankiniu būdu. Todėl „MIMOSA“ – tai tik priemonė patogiau dirbti, greičiau atlikti tam tikrus veiksmus, priimti sprendimus, daryti pakeitimus ar ištaisyti klaidas. Dirbant labai svarbu žinoti tvarkaraščio sudarymo taisykles ir subtilumus. Daugiausiai laiko užimantis darbas – duomenų suvedimas.

Kita mokymo įstaigų užsiėmimų tvarkaraščių sudarymo sistema yra pamokų tvarkaraščių sudarymo sistema „REKTOR“ (Smykalov, 2001). Programos pagrindinė idėja yra panaši į „MIMOSA“ programos idėją. Tvarkaraščių sudarinėtojas turi tvarkaraščius sudėlioti ir derinti pats. Tai tampa labai sudėtingu procesu, kai yra daug mokinių ir atsiranda nors menkiausi pakeitimai. Sistemoje „REKTOR“ yra sukurta labai lanksti ir patogi grafinė sąsaja. Programa yra išleista lietuvių kalba ir nėra taip sudėtingai valdoma kaip „MIMOSA“. Ši programa reikalauja apie 75% žmogiškųjų išteklių.

Dar viena užsiėmimų tvarkaraščių sudarymo sistema – Slovakijos IT kompanijos Applied Software Consultants s.r.o. produktas „aSc Tvarkaraščiai“ (aSc TimeTables, 2007). Galinga ir kol kas vienintelė rinkoje tvarkaraščių kūrimo programa lietuvių kalba, kuri sugeba sukurti visos mokyklos tvarkaraštį, į jį sudėdama absoliučiai visas reikiamas pamokas, jei tiktai tam yra galimybė, patikrindama tūkstančius įvairiausių galimų variantų. Šia programa galima greitai ir paprastai sukurti mokyklos, gimnazijos, profesinės mokyklos, licėjaus, aukštosios mokyklos ar kitos mokslo įstaigos tvarkaraščius, kurie atitiks visus organizacinius, psichologinius bei pedagoginius reikalavimus. Tvarkaraščiai gali būti kiek vienasavaitiniai, tiek daugiasavaitiniai.

Programa turi galingą algoritmą, leidžiantį jai sudaryti tvarkaraštį per keletą minučių, sudedant į tvarkaraštį visas pamokas ir užsiėmimus. Vėliau galima tvarkaraštį redaguoti, keičiant sąlygas ar net rankiniu būdu išdėstant pamokas. Jeigu pasirodo, kad tvarkaraščio sukurti neįmanoma, programa turi testavimo įrankius, leidžiančius surasti problemą, ar tai būtų įvestų duomenų klaida, ar pernelyg griežtos jai užduotos sąlygos. Ši programa reikalauja apie 20-30% žmogiškųjų išteklių.

Mokyklos pamokų tvarkaraščio optimizavimo programa „School – COMPLETE“ sugeba optimizuoti nusiųstą užsiėmimų tvarkaraštį, sumažindama „langų“ skaičių. Tvarkaraštis turi būti užkoduotas specialiu duotu formatu, nusiųstas programai, kuri pateiks optimizuotą tvarkaraščio variantą. Programa naudoja keletą paprastų optimizavimo algoritmų (School–Complete). Labiausiai programa pritaikyta pagrindinių mokyklų pamokų tvarkaraščiui optimizuoti. Yra sukurta patobulinta „Mokyklos tvarkaraščių optimizavimo sistema“ („School schedule optimization program“). Ši sistema dabar yra sukurta anglų kalba, tačiau turi labai išsamią pagalbą vartotojui lietuvių kalba. Pateikus programai pradinius duomenis, mokyklos tvarkaraščių optimizavimo sistema atlieka pradinį profiliuotų klasių tvarkaraščio sudarymą. Po to tvarkaraščių sudarinėtojas gali nustatyti optimizavimo kriterijus pagal tam tikros mokyklos reikalavimus. Atliekamas pradinio tvarkaraščio optimizavimas įgalina iki minimumo sumažinti „langų“ skaičių tiek mokiniams, tiek mokytojams. Kadangi kiekvienas mokinys gali pasirinkti jam patinkančius dalykus, tai tvarkaraštis turi būti sudaromas individualiai kiekvienam mokiniui. Tačiau tvarkaraštis turi būti priimtinas ne tik mokiniams, bet ir mokytojams. Turi būti išpildyti ir kai kurie fiziniai apribojimai (pasirinkti optimizavimo kriterijai). Sukurta programa palengvina profilinių klasių tvarkaraščio sudarymą. Vartotojas dalyvauja tik duomenų failo sudaryme, duomenų failo nusiuntime į serverį ir rezultatų gavime. Programa sudaro tvarkaraštį pagal nurodytus apribojimus, atlieka optimizavimą, kiekvienam tvarkaraščio variantui apskaičiuoja baudos taškus ir išveda geriausią tvarkaraštį, gautą pagal baudos taškų skaičių. Programa sukurta naudojant java kalbą. Su šia programa gali būti dirbama tiek personaliniame kompiuteryje, tiek internetu.

Kita kategorija – transporto tvarkaraščių sudarymo programinė įranga. Transporto tvarkaraščių programa „Pikas“, kurią kuria kompanija „Merakas“, naudojasi Baltijos ir kai kurios kitos užsienio šalys. Tai labai lanksti programa, leidžianti sistemiškai sutvarkyti transporto grafikus, koordinuoti keleivių srautus ir suderinti maršrutus („Pikas“ – maršrutinio transporto darbo planavimas, 2007). Norint pasiekti optimalų variantą, reikėtų pagal šią sistemą suderinti viso miesto transporto tvarkaraščius (autobusų ir troleibusų) bei įvesti tam tikrą laiko tarpą galiojančius bilietus, kad keleiviai galėtų perlipti iš vienos transporto priemonės į kitą. Ta pati kompanija kuria dar du programinius produktus transporto srautams valdyti – „Mobis“ ir „Ratas“.

Dar viena kategorija – projektavimo tvarkaraščių sudarymo programinė įranga. Project Standard 2003 yra integruota Microsoft Office sistemos dalis, todėl vienu metu galima naudotis kitais produktais, pvz., Microsoft Office PowerPoint® 2003 ir Microsoft Office Visio® 2003, ir taip

efektyviai pateikti esamos būsenos projektą (Project Standard 2003 apžvalga, 2003). Šis programinis produktas yra skirtas projektams planuoti ir valdyti. Naudojant Microsoft Office Project Standard 2003, galima efektyviai vykdyti ir sekti savo projektuose atliekamas užduotis ir juose išnaudojamus išteklius, kad projektai visuomet būtų pateikti laiku ir neišeikvotų papildomų lėšų. Kuriant tvarkaraščius galima sekti ir įvertinti įvairius planus ir jų alternatyvas, numatytos galimybės nustatyti kainas ir atitinkamais projekto valdymo įrankiais sekti projekto pažangą. Užduotims atlikti galima priskirti išteklius taip, kad išvengtume nesuderinamumo bei tų pačių išteklių pakartotinų paskirstymų. Project Standard 2003 gali automatiškai atnaujinti planus, kad būtų galima efektyviai nustatyti darbų prioritetus ir priimti geresnius sprendimus. Galima sekti pažangą, nuolat peržiūrėti planuojamus ir faktinius projekto tikslus bei naudotis senesniais įrašais.

Komercinis produktas ResSched™ – tvarkaraščių sudarymo programinė įranga bei vadybos įrankis verslo, vyriausybinėms, mokymo, sveikatos apsaugos ir kitoms organizacijoms (ResSched – Scheduling Software and Management Tools, 2006). ResSched naudojama šimtuose organizacijų tam, kad sudaryti tvarkaraščius bei valdyti personalą, patalpas, įrangą, prietaisus, transportą ir kitus resursus.

Yra sukurtų ir atvirojo kodo sistemų. Viena iš jų – Job Scheduler (Open Source - Job Scheduler, 2005). Pagrindinės savybės – galima spręsti sudėtingus tvarkaraščių sudarymo uždavinius, pvz. su sąlyginėmis darbų pradžiomis, numatytas lygiagretus darbų vykdymas, užduočių srautų kompiuterinėse sistemose valdymas, leidžia kurti ir operuoti darbų grandinėmis.

Taip pat yra sukurta įvairių programų paketų, kuriuose yra įtraukti atskiri komponentai, skirti įvairiems tvarkaraščių uždaviniams spręsti ar su tvarkaraščiais susijusiems procesams modeliuoti. Programų paketas WinQSB skirtas įvairiems otimizavimo bei operacijų tyrimo uždaviniams spręsti (MathSoft WinQSB, 2003). Jame galima rasti daug įvairių įrankių (programinių modulių), skirtų tiesinio, kvadratinio, netiesinio, sveikaskaičio programavimo uždaviniams, taip pat sprendimų priėmimo, darbų tvarkaraščių sudarymo bei kitiems specifiniams uždaviniams spręsti. WinQSB moduliai gali būti sėkmingai taikomi vadybos, verslo uždaviniams spręsti. Visi programiniai moduliai turi patogias ir suprantamas vartotojo sąsajas, pradiniai duomenys dažniausiai pateikiami lentelių pavidalu. Rezultatai pateikiami lentelėmis ir diagramomis. Viena iš WinQSB paketo programinių modulių grupių yra specialiosios paskirties moduliai, kuriais galima spręsti tik tam tikros klasės optimizavimo uždavinius (pvz. darbų tvarkaraščių sudarymo uždavinius). Galima išskirti šiuos WinQSB paketo specialiosios paskirties modulius: Project Scheduling: PERT-CPM (Program Evaluation and Review Technique – Critical Path Method) – kalendorinio planavimo (tvarkaraščių) uždaviniams spręsti, Queuing Analysis (QA), Queuing System Simulation (QSS) – masinio aptarnavimo (eilių) sistemų analizės ir modeliavimo uždaviniams spręsti, Material Requirements Planning (MRP) – įvairių išteklių poreikių planavimo uždaviniams spręsti, Job Scheduling (JOB) – darbų tvarkaraščių sudarymo uždaviniams spręsti. Visiems programiniams

moduliams sukurtos vartotojo sąsajos yra paprastos, patogios ir pritaikytos specifiniams uždaviniams.

Galingą imitacinio modeliavimo programų paketą ARENA sukūrė komapanija „Rockwell Software“ (Simulation with ARENA, 2007). „Arena“ yra viena iš žinomiausių ir labiausiai taikomų imitacinio modeliavimo sistemų. Ši programinė įranga yra pasaulinis lyderis modeliavimo srityje, sėkmingai naudojama įvairių pasaulio organizacijų verslo efektyvumui ir produktyvumui gerinti bei kitose srityse. „Arena“ turi lengvai suprantamas įvairių programinių modulių vartotojo sąsajas, labai vaizdžias įvairių procesų vizualizavimo priemones, rezultatai gali būt išsamiai analizuojami. Profesionalioji „Arena“ versija efektyviausia tada, kai reikia analizuoti sudėtingus, vidutinės ar didelės apimties projektus, kurie ypač „jautrūs“ pasikeitimams, dažniausiai taikoma įvairioms tiekimo grandims analizuoti, gamybos procesų, logistikos, paskirstymo, sandėliavimo ir aptarnavimo sistemoms modeliuoti. Taip pat yra numatyti įvairūs specifiniams taikymams ar pramonei skirti šablonai, galima kurti savus šablonus, kuriuos sudaro modeliuojamųjų objektų bibliotekos. Vėliau sukurtuosius šablonus galima panaudoti panašaus pobūdžio uždaviniams spręsti. Tačiau sistema labiau skirta modeliuoti įvairiems aptarnavimo procesams ir joje taikomi paprasčiausi perrinkimo būdai racionaliems (optimaliems) sprendiniams surasti.

2.6 Išvados

Tvarkaraščių uždavinių analitinis tyrimas leidžia išskirti tokias pagrindines šių uždavinių savybes.

1. Didžioji dalis praktinių tvarkaraščių uždavinių yra pakankamai sudėtingi ir klasikiniais metodais galima gauti tik leistiną sprendinį, kuris nebus optimalus. Dažniausiai uždavinių apimtys būna didelės, ribojimų labai daug ir perrinkti visus galimus variantus tampa neįmanoma, todėl tenka ieškoti kitų metodų, besiremiančių įvairiomis euristinėmis paradigmomis.
2. Testuojant sukurtus algoritmus bei tiriant jų efektyvumą yra pasinaudojama standartinių testinių pavyzdžių duomenų bazėmis, ir apie sukurtų algoritmų privalumus yra sprendžiama lyginant didelio skaičiaus testinių uždavinių sprendimo rezultatus.
3. Tvarkaraščiams įvairiose srityse sudaryti naudojama programinė įranga dažniausiai remiasi interaktyviu dialogu ir reikalauja daug „rankinio darbo“.
4. Tvarkaraščiams sudaryti yra taikomi paprasčiausi optimizavimo algoritmai. Sudėtingi šiuolaikiniai metaeuristiniai algoritmai standartinėse programinėse sistemose beveik nėra diegiami.

3 skyrius. Tvarkaraščių su ribotais ištekliais formalizavimas

3.1 Tvarkaraščių su ribotais ištekliais uždavinio formuluotė

Inžinerinio projektavimo ir vadybos uždaviniuose dažnai susiduriama su įvairiomis darbų arba užduočių tvarkaraščių sudarymo bei kalendorinio planavimo problemomis, kai užduotys turi būti atliktos per tam tikrą laiką, naudojant tam reikalingus išteklius. Ištekliai bei laikas, reikalingas projektui įgyvendinti, gali būti riboti. Reikia rasti užduočių atlikimo tvarkaraštį, tenkinantį nuoseklumo sąryšius, išteklių ribojimus, minimizuojant tam tikrus kriterijus. Tokiu kriterijumi gali būti viso projekto įgyvendinimo laikas, sąnaudos, reikalingos projektui atlikti ir pan. Ši uždavinių klasė mokslinėje literatūroje vadinama tvarkaraščių su ribotais ištekliais uždaviniais (Ribotų išteklių tvarkaraščių sudarymo uždaviniai, RITSU; *angl.,* Resource Constrained Project Scheduling Problems, RCPSP).

Tokių uždavinių pradiniai duomenys yra tam tikros užduočių, susijusių nuoseklumo sąryšiais, aibės formalizuotas aprašymas. Detaliau panagrinėkime šį aprašymą. Pažymėkime $J = \{0,1,...,n,n+1\}$ užduočių indeksų aibę, *0-inė* ir *(n+1)-oji* užduotys yra fiktyvios (*dummy*) ir reiškia viso projekto pradžią ir pabaigą, $p_j,\ j \in J$ – *j*-osios užduoties trukmė, $p_j \geq 0,\ j \in J$ – sveikieji neneigiami skaičiai ir $p_0 = p_{n+1} = 0$.

Nuoseklumo sąryšius aibėje *J* nusakysime porų aibe $C = \{ (i, j) \mid i \text{ vykdomas prieš } j \}$. Pažymėkime $K = \{1,...,m\}$ aibę išteklių indeksų. Visi ištekliai laikomi atnaujinamais (renewable) ir nesudedamais – kiekvienu laiko momentu išskiriama fiksuota kiekvieno tipo išteklių apimtis, o jų likučiai dingsta. Laikysime, kad išskiriama atskiros rūšies išteklių apimtis $R_k > 0, k \in K$, yra pastovus dydis. Galime pažymėti $r_{jk} \geq 0$ *k*-tųjų išteklių apimtį (sąnaudas), kurios reikia *j*-ajai užduočiai atlikti. *j*-osios užduoties atlikimo pradžios momentas yra $s_j \geq 0$. Tariame, kad užduotys vykdomos be pertraukų ir *j*-osios užduoties pabaigos momentas nusakomas lygybe $c_j = s_j + p_j$.

Užduočių, atliekamų laiko momentu *t*, indeksų aibė $A(t) = \{ j \in J \mid s_j \leq t < c_j \}$. Projekto užbaigimo laiką pažymėkime *T(S),* $T(S) = c_{n+1}$.

Įvedę šiuos pažymėjimus, užrašysime minimizavimo uždavinio formuluotę.

Rasti $\min T(S)$, atsižvelgiant į šias sąlygas:

$$c_i \leq s_j, (i, j) \in C, \tag{3.1}$$

$$\sum_{j \in A(t)} r_{kj} \leq R_k,\ k \in K,\ t \geq 0, \qquad s_j \geq 0,\ j \in J. \tag{3.2}$$

Tikslo funkcija nusako viso užduočių komplekso užbaigimo laiką. (3.1) nelygybė nusako nuoseklumo sąryšius, (3.2) reikalauja paisyti išteklių ribojimų.

Šis uždavinys yra NP sudėtingumo stipriąja prasme, kadangi vienmatis pakavimo į konteinerius uždavinys yra šio uždavinio atskiras atvejis (Kocetov, Stoliar, 2003). Juolab, bet kuriam $\varepsilon > 0$, mažai tikėtina rasti apytikslius polinominius algoritmus su garantuotu tikslumo įverčiu $n^{1-\varepsilon}$. Todėl euristikų kūrimas yra perspektyviausias kelias tokių uždavinių sprendimui.

3.2 RITSU duomenų formalizavimas ir prielaidos

Tvarkaraščio formalizavimas remiasi keliomis prielaidomis, kurias dabar aptarsime. Tvarkaraščio uždavinys yra suvedamas į darbų pradžių (darbų pabaigų) vektoriaus nustatymą remiantis kitais pradiniais duomenimis. Pradiniai duomenys, apibūdinantys tvarkaraštį, yra šie:

- Užduočių nuoseklumo saryšių matrica,
- Užduočių trukmių vektorius,
- Išteklių, reikalingų užduotims atlikti, matrica,
- Išteklių ribojimų vektorius.

7.1 skyrelyje bei D priede galima rasti formalizuoto tvarkaraščio aprašymo duomenų pavyzdžius.

Paprastai yra laikoma, kad tvarkaraštis yra sudarytas, jei yra žinomi užduočių pradžių vektoriai. Pasinaudojant užduočių pradžių vektoriais yra nustatomi darbų pabaigų vektoriai. Kartais, pirmiau yra nustatomi darbų pabaigų vektoriai, o pagal juos yra apskaičiuojami darbų pradžių vektoriai. Pagal darbų pradžių ir pabaigų vektorius yra nustatomi ankstyvieji ir vėlyvieji darbų pradžios ir pabaigos laikai. Sudaryto tvarkaraščio atskirų darbų ankstyvieji ir vėlyvieji darbų pradžios ar pabaigos laikai nustato laiko intervalą, kuriame galima pakeisti tų darbų pradžios ar pabaigos laiką, nepakeitus kitų darbų pradžios ar pabaigos laikų.

Užduotys yra apibrėžiamos kaip nedalomi veiklos vienetai, kurie negali būti dalomi į mažesnius vienetus. Pradėta užduotis negali būti pertraukiama ir turi būti vykdoma iki galo. Jei vykdomo metu užduotys gali pereiti kelias skirtingas vykdymo stadijas, tai tvarkaraštis vadinamas daugelio būsenų (multi-mode). Atitinkamai, jei vykdymo metu užduočių būsenos nesikeičia, tvarkaraštis vadinamas vienos būsenos (single-mode). Šiame darbe apsiribosime vienos būsenos tvarkaraščiais.

Kiekvienu (vienos būsenos) projekto įgyvendinimo momentu užduotis gali būti:

- **neparengta** (tuo momentu ji negali būti pradėta vykdyti, nes nepatenkinti tam tikri kiti reikalavimai),
- **parengta** (t.y. ją jau galima vykdyti),
- **vykdoma** (kai tam tikras vykdytojas ją jau atlieka) arba
- **įvykdyta.**

Su kiekviena užduotimi yra siejamas skaičius, vadinamas užduoties *vykdymo trukme*. Jis reiškia nepertraukiamą laiko intervalą, per kurį užduotis yra atliekama. Užduotis žymėsime didžiosiomis raidėmis *A*, *B, C* ir t.t., nors kartais patogumo dėlei vartojamos santrumpos (pavyzdžiui, *SD* – stogo dengimas, *EI* – elektros instaliacija ir pan.). Skliausteliuose po užduoties simbolio yra įrašoma jos

trukmė. Pavyzdžiui, *A(5)* reikš, kad užduoties *A* vykdymas truks 5 laiko vienetus (minutes, valandas ar kita), ir nesvarbu, ar tai daro V_1, V_2, ar dar kuris kitas vykdytojas.

Nuoseklumo sąryšiai nusako apribojimus užduočių vykdymo tvarkai. Būdinga nuoseklumo nurodymo forma yra tokia: *užduotis X eina prieš užduotį Y,* ir tai reiškia, kad užduotis Y negali būti pradėta vykdyti tol, kol neįvykdyta užduotis *X.* Tokią tvarką patogu žymėti $X \rightarrow Y$. Jei nei $X \rightarrow Y$, nei $Y \rightarrow X$, tai sakysime, kad užduotys X ir *Y* yra **nepriklausomos.** Yra ir tokių tvarkaraščių sudarymo uždavinių, kuriuose visos užduotys nepriklausomos (žiūr. 4.1.3 skyrelį).

Nuoseklumo sąryšiai pasižymi tokiomis savybėmis:

- nuoseklumo sąryšiai yra *tranzityvūs:* jei $X \rightarrow Y$ ir $Y \rightarrow Z$, tai turi būti teisinga, kad $X \rightarrow Z$.
- nuoseklumo sąryšiai *neturi sudaryti ciklo*
  (negali būti tokių sąryšių sekų: $X \rightarrow Y$, $Y \rightarrow Z$ ir $Z \rightarrow X$).

Jei nuoseklumo sąryšių aprašyme pasitaiko ciklų, reiškia duomenys yra nekorektiški.

3.3 Grafų teorijos taikymas tvarkaraščiams formalizuoti

3.3.1 Tvarkaraščių vaizdavimas orgrafais

Tvarkaraščius patogu vaizduoti kryptiniais grafais (orientuotaisiais, orgrafais). Priede A yra pateikta grafų teorijos sąvokų, taikomų tvarkaraščių teorijoje, santrauka.

3.1 pavyzdys. Panagrinėkime būsto statybos (BS) uždavinį, kai dirba keli vykdytojai. BS uždavinį sudaro 15 skirtingų užduočių. Jų pavadinimai ir vykdymo trukmės valandomis (ar dienomis) nurodytos 3.1 lentelėje. BS uždavinio sąlygoje nurodyta 17 skirtingų užduočių nuoseklumo sąryšių, kurie išvardyti 3.2 lentelėje.

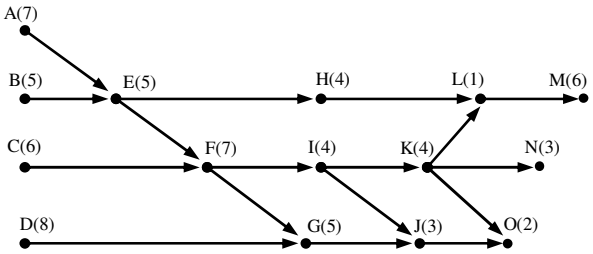
3.1 lentelė. Užduotys ir jų trukmės

Užduotys	Žymėjimas
Pamatų klojimas	*A(7)*
Grindų surinkimas	*B(5)*
Sienų surinkimas	*C(6)*
Kupolo surinkimas	*D(8)*
Grindų tvirtinimas	*E(5)*
Vidinių sienų tvirtinimas	*F(7)*
Kupolo tvirtinimas	*G(5)*
Vandentiekio įrengimas	*H(4)*
Jėgainės įrengimas	*I(4)*
Elektros instaliacija	*J(3)*
Šildymo prietaiso įrengimas	*K(4)*
Kanalizacijos įrengimas	*L(1)*
Vidinės apdailos darbai	*M(6)*
Sandarinimas	*N(3)*
Komunikacijų patikrinimas	*O(2)*

3.2 lentelė. Sąryšiai

Nuoseklumo sąryšiai		
A	→	*E*
B	→	*E*
E	→	*F*
C	→	*F*
D	→	*G*
F	→	*G*
E	→	*H*
F	→	*I*
I	→	*J*
G	→	*J*
I	→	*K*
H	→	*L*
K	→	*L*
J	→	*O*
K	→	*O*
L	→	*M*
K	→	*N*

Šiuos duomenis galima atvaizduoti grafu, pateiktu 3.1 pav.



3.1 pav. Pradinių duomenų atvaizdavimas grafu

Grafas, vaizduojantis tvarkaraščio duomenis, kartais yra vadinamas tinkliniu grafiku. Todėl tvarkaraščių sudarymo būdai, besiremiantys tokiu vaizdavimu, yra vadinami tinkliniu planavimu (projektavimu).

Pasinaudojant orgrafais galima vaizduoti ne tik užduočių nuoseklumo sąryšius, bet taip pat jų trukmes bei išteklių sąnaudas, reikalingas užduotims atlikti. Sutinkami du vaizdavimo būdai – „užduotys-viršūnėse“ (acitivity-on-node) ir „užduotys-briaunose“ (activity-on-branch) (Glazman, Novikov, 1966, Viliūnas, 1976, Tanajev, Kovaliov, Šafranskij, 1998, Hendrickson, 2000).

3.3.2 Užduočių nuoseklumo sąryšių, jų trukmių ir reikiamų išteklių vaizdavimas „užduotys-viršūnėse“ būdu

Panagrinėkime tvarkaraščio duomenų vaizdavimą pagal „užduotys-viršūnėse“ būdą. Čia kiekvieną užduotį atitinka grafo viršūnė, vaizduojama kvadratu, kuriame įrašyti užduoties parametrai:

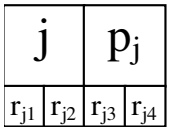
j – užduoties numeris,

p_j – užduoties *j* vykdymo laikas,

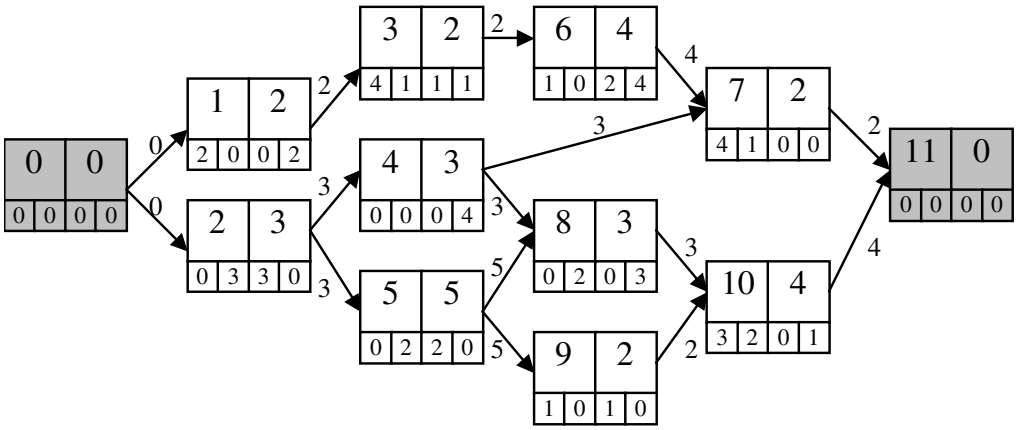
r_{jk} – *k*-osios rūšies išteklių sąnaudos *j-qjai* užduočiai atlikti.

Užduočių nuoseklumo saryšiai tvarkaraštyje gali būti nusakomi necikliniu kryptiniu grafu $G=(V,A)$. $V=\{1,\ldots,n\}$ – užduočių aibė. Grafo lankai nurodo užduočių atlikimo nuoseklumą (žiūr. pavyzdį 3.3 pav.). Bendru atveju turimos išteklių apimtys kiekvienu laiko momentu gali būti kintamos. Nuoseklumo sąryšių grafe dar gali būti nurodomi intervalai nuo *i*-osios užduoties pradžios iki *j*-osios užduoties pradžios (galimi ir variantai pradžia-pabaiga, pabaiga-pradžia, pabaiga-pabaiga). Kai šie intervalai yra didesni, nei *i*-osios užduoties trukmė, tai reiškia, jog yra nusakyti laukimo (prastovos) laikai. Uždavinio pradinėse sąlygose būna duotos išteklių R_k apimtys $(k = 1,...,m)$, kurių viršyti negalime viso projekto vykdymo metu, pavyzdžiui, $R_k = 4$, $k=1,...,4$.

Apibendrintu atveju užduočių nuoseklumo sąryšiai dar gali turėti parametrą λ_{ij}, kuris yra minimalus atstumas (laukimo laikas) tarp užduoties *i* baigimo momento ir užduoties *j* pradžios momento. Mūsų nagrinėjamu atveju $\lambda_{ij} = 0$, visiems $i, j = 1,\ldots n$.



3.2 pav. Užduočių nuoseklumo grafo viršūnės duomenys



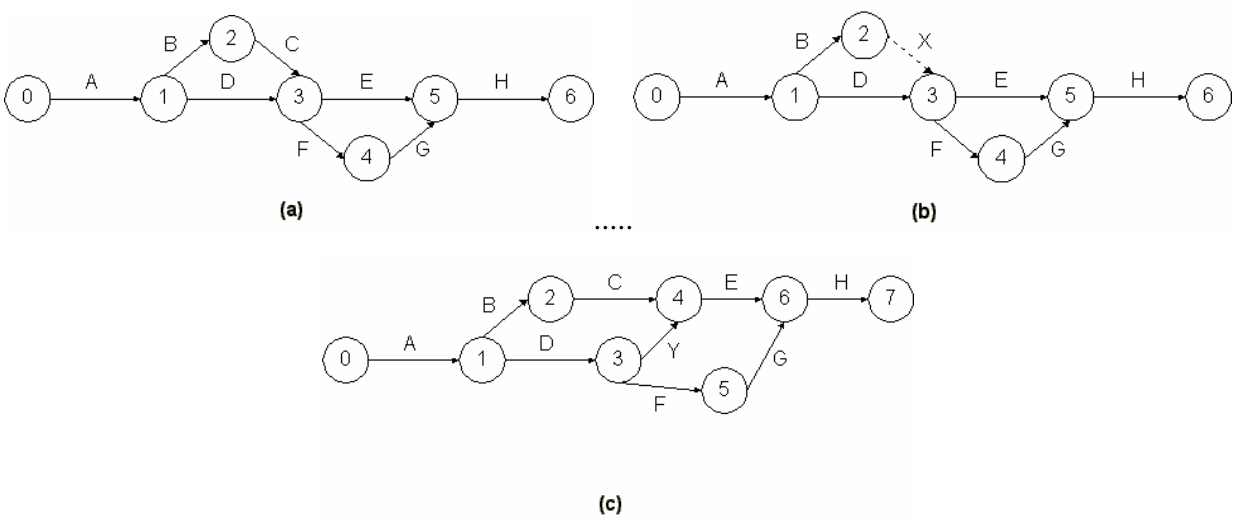
3.3 pav. Užduočių nuoseklumo sąryšių grafas bei duomenys apie užduotis (*n*=10, *m*=4, *R*=4)

3.3 pav. pateiktas nuoseklumo sąryšių grafo pavyzdys, kai užduočių skaičius $n=10$, užduotys 0 ir 11 – fiktyvios, išteklių rūšių skaičius $m=4$, kiekvieno iš išteklių viršutinė riba $R=4$.

3.3.3 Tvarkaraščio pradinių duomenų vaizdavimas „užduotys-briaunose“ būdu

Grafinis tvarkaraščio duomenų vaizdavimas vietoj paprasto sąrašo yra labai naudingas ir intuityviai suprantamas, taip pat galima lengviau įsitikinti, jog visi ribojimai tenkinami (būna, jog programose duomenų įvedimas atliekamas įrašant pradinius duomenis tiesiog į tuščius laukus grafe).

Naudojat „užduotys-briaunose“ (activity-on-branch) vaizdavimo būdą, pačios užduotys vaizduojamos kaip briaunos ar ryšiai tarp viršūnių. Pačios viršūnės – tai lyg įvykiai arba gairės, apibrėžiančios galimas užduočių pradžias ir pabaigas. Jei naudojamas šis vaizdavimo būdas, fiktyvios užduotys gali būti papildomai įvedamos tam, kad išlaikyti reikiamą užduočių nuoseklumą. Laikoma, jog fiktyvi užduotis neturi trukmės ir grafe gali būti pavaizduota punktyrine linija. Keli atvejai, kai fiktyvias užduotis verta įvesti, yra pavaizduoti 3.4 pav. Jei 3.4 pav. (a) atveju pašalinsime užduotį C, tai reikš, jog užduotys B ir D yra tarp viršūnių 1 ir 3. Tačiau, jeigu įvesime fiktyvią užduotį X, kaip parodyta 3.4 pav. (b), pradiniai reikalavimai užduotims B (tarp viršūnių 1 ir 2) ir D (tarp viršūnių 1 ir 3) bus išlaikyti. Be to, jei pradinius duomenis atveju (a) pakeistume taip, jog užduotis E negali būti pradėta, kol nebaigtos užduotys C ir D, o užduotis F gali būti pradėta, kai tik bus pabaigta užduotis D, naują užduočių atlikimo tvarką galima būtų pavaizduoti pridedant naują fiktyvią užduotį Y, kaip parodyta (c) atveju. Taigi, fiktyvus užduočių įvedimas tokiais atvejais būtinas, norint patenkinti specifinio uždavinio reikalavimus.



3.4 pav. Fiktyvios užduotys projekto grafe

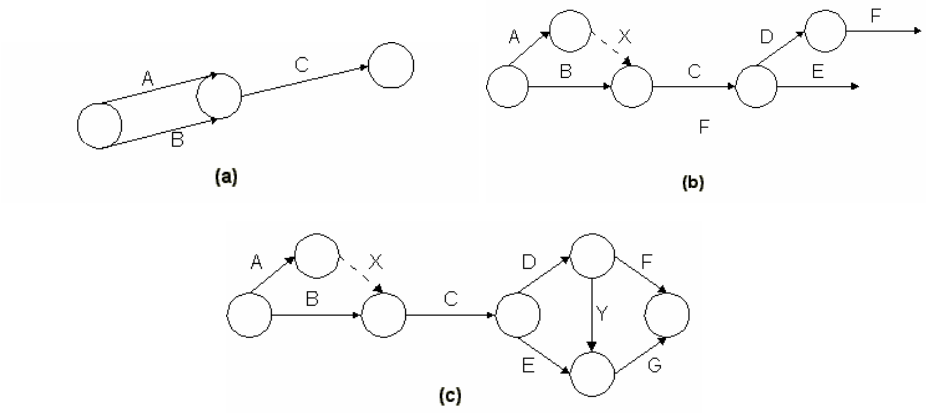
Daugelis tvarkaraščių modeliavimo sistemų palaiko tik vieną užduočių tinklo vaizdavimo grafu būdą – arba „užduotys-briaunose“ (activity-on-branch), arba „užduotys-viršūnėse“ (acitivity-on-node).

3.2 pavyzdys. Tarkime, jog norime grafu pavaizduoti veiksmų nuoseklumą, kai turime septynias užduotis, susijusias nuoseklumo sąryšiais. Šie sąryšiai pateikti 3.3 lentelėje.

3.3 lentelė. Užduočių ir jų nuoseklumo sąryšių pavyzdys

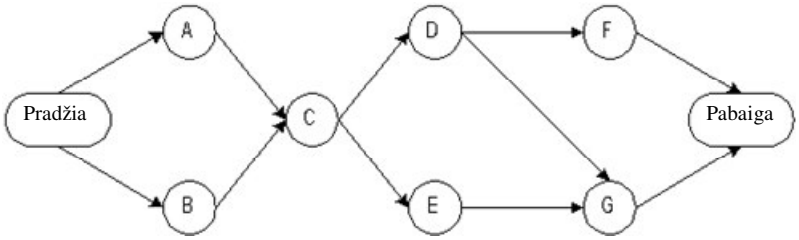
Užduočių pavadinimai	Užduotys, kurios turi būti atliktos prieš kiekvieną iš užduočių
A	---
B	---
C	A,B
D	C
E	C
F	D
G	D,E

Formuoti grafą „užduotys-briaunose“ principu šiai užduočių aibei reiktų pradėti brėžiant briaunas užduotims A, B ir C, kaip parodyta 3.5 pav. (a). Pažymėtina, jog dvi užduotys (A ir B) yra tarp tų pačių viršūnių, žyminčių įvykius. Dėl aiškumo, įterpiame fiktyvią užduotį X (jis įpareigos prieš pradedant užduotį C sulaukti užduočių A ir B pabaigos) bei brėžiame briaunas kitoms užduotims, kaip parodyta 3.5 pav. (b). Užduoties G atvaizdavimas tampa problema, jei norime, jog prieš šią užduotį būtų atliktos užduotys D ir E. Todėl pridedame fiktyvią užduotį Y, kuri kartu su G užbaigia tinklo formavimą, kaip parodyta 3.5 pav. (c).



3.5 pav. Tinklo formavimas „užduotys-briaunose“ principu

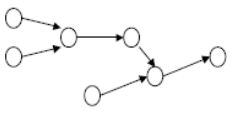
Palyginimui 3.6 pav. pateiktas atvaizdavimas „užduotys-viršūnėse“ principu. Šiuo atveju pridėtos fiktyvios projekto pradžios (start) ir pabaigos (finish) viršūnės. Pažymėtina, jog šiuo vaizdavimo atveju fiktyvios užduotys įvedamos ne tam, jog išreikštų nuoseklumo sąryšius, bet fiksuoti viso projekto pradžią ir pabaigą.



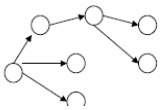
3.6 pav. Tinklo formavimas „užduotys-viršūnėse“ principu

3.3.4 Nuoseklumo sąryšių tipai

Grafiškai vaizduojant užduočių nuoseklumą, galima susidurti su keletu sąryšių tipų. Vienas iš tokių tipų – grandinės. Grandinės (chains) – kai prieš kiekvieną užduotį turi būti atlikta ne daugiau kaip viena užduotis bei ne daugiau kaip viena užduotis turi būti atlikta po jos. Atvejis, kai po kiekvienos užduoties seka ne daugiau kaip viena kita užduotis (intree), pavaizduotas 3.7 pav., o atvejis, kai prieš kiekvieną užduotį turi būti atlikta ne daugiau kaip viena užduotis (outree), pavaizduotas 3.8 pav.



3.7 pav. Sąryšiai „intree“

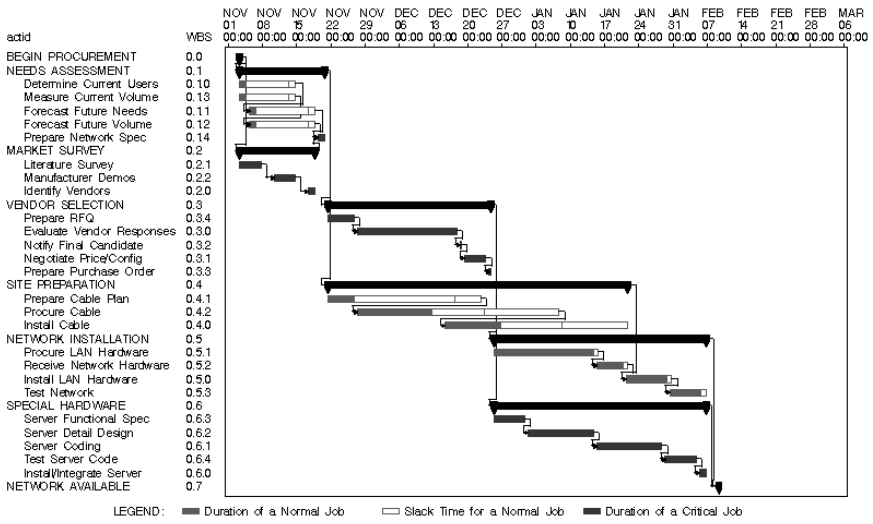


3.8 pav. Sąryšiai „outree“

3.4 Laiko diagramos

Vykdymo metu tvarkaraščius patogu vaizduoti laiko diagramomis (Gannt chart). Darbų diagramą sudaro vertikalus užduočių sąrašas, kur kiekvienos užduoties trukmė vaizduojama

horizontaliu brūkšniu (ar stačiakampiu), kurio pradžia ir pabaiga atitinka užduoties pradžios ir pabaigos momentus.



3.9 pav. Darbų (Gannt chart) diagrama

3.5 Klasikinio RITSU prielaidos

Klasikinis RITSU yra formuluojamas, remiantis šiomis prielaidomis:

- nuoseklumo sąryšiuose užduočių vėlinimo laikai lygūs 0,
- nė viena užduotis negali būti vykdoma, kol neįvykdytos visos prieš ją būtinos įvykdyti užduotys,
- kiekviena užduotis *j* turi savo fiksuotą trukmę p_j,
- kiekviena užduotis *j* naudoja r_{jk} – fiksuotą *k*-ųjų išteklių vienetų skaičių *r*,
- kiekvienų *k*-ųjų išteklių apimtis R_k yra fiksuota,
- užduoties vykdymas negali būti nutrauktas jos nebaigus,
- užduočių paleidimo (pradžios) ar baigimo laikai nėra numatyti iš anksto.

Pastarieji laikai yra randami minimizuojant viso projekto atlikimo laiką, atsižvelgiant į išteklių ribojimus ir nuoseklumo sąryšius.

Projekto pradiniai duomenys yra vaizduojami grafu (tinkliniu grafiku), pavaizduotu 3.3 pav. Tvarkaraščio planavimo metu kiekvienai užduočiai apskaičiuojami ES_j (ankstyvasis pradžios laikas) ir LF_j (vėlyvasis baigimo laikas):

$ES_j = max\ \{EF_i \mid i \in P_j\},\ EF_j = ES_j + p_j;\ j = 1, \ldots, n+1$

$LF_j = min\ \{LS_h \mid h \in F_j\},\ LS_j = LF_j - p_j;\ j = n, \ldots, 0$

Tada intervalas $[ES_j\ ,\ LF_j]$ apibrėžia laiko tarpą, kuriame turi būti vykdoma užduotis *j*. Ši užduotis negali būti pradėta anksčiau, nei laiko momentas ES_j, nepažeidžiant nuoseklumo sąryšių. Jei užduotis bus pradėta vėliau, nei LF_j, tai viso projekto vykdymo laikas padidės.

3.6 Apibendrinto RITSU prielaidos

Apibendrinto RITSU atveju užduočių nuoseklumas gali priklausyti nuo laukimo laiko, skirto užduotims paruošti. Atskiru atveju gali būti nusakyti užduočių prioritetai, kurie įtakotų reikalavimą nutraukti vykdomą užduotį, kad būtų galima atlikti aukštesnio pirmumo užduotį.

Bendru atveju gali būti reikalaujama, kad tarp užduočių būtų tam tikri laiko tarpai arba užduotys prasidėtų ne anksčiau arba ne vėliau nustatyto laiko. Yra išskiriami 4 skirtingi nuoseklumo sąryšių tipai, aprašantys užduočių vėlinimo laiką:

$SS_{ij} \geq 0$ – minimalus laiko tarpas tarp užduočių *i* ir *j* pradžių (start-to-start),

$FF_{ij} \geq 0$ – minimalus laiko tarpas tarp užduočių *i* ir *j* pabaigų (finish-to-finish),

$FS_{ij} \geq 0$ – minimalus laiko tarpas tarp užduoties *i* pabaigos ir užduoties *j* pradžios (finish-to-start),

$SF_{ij} \geq 0$ – minimalus laiko tarpas tarp užduoties *i* pradžios ir užduoties *j* pabaigos (start-to-finish).

Kitaip nei klasikiniu atveju, ribojimai gali būti kiekvienos užduoties pradžios laikui ρ_j ir baigimo laikui δ_j. Pradžios laikas ρ_j nusako momentą, už kurį užduotis j negali būti pradėta anksčiau. Baigimo laikas δ_j nustato laiko ribojimą, kada reiktų baigti užduoties vykdymą. Išteklių apimtys R_{kt} gali būti kintamos, pvz. R_{kt} nusako *k*-ųjų išteklių esamas apimtis laiko intervale (t-1, t].

Nuoseklumo sąryšių aprašas šiuo atveju gali būti supaprastintas, naudojant sveikuosius skaičius λ_{ij}, kurie nusako laiko langus start-to-start, ir gali būti iš kitų išreiškiami taip:

$\lambda_{ij} = SF_{ij} - p_j$; $\lambda_{ij} = FS_{ij} + p_i$; $\lambda_{ij} = FF_{ij} + p_i - p_j$;

3.7 Ribojimai procesorių skaičiui

Procesoriumi (mašina) galime vadinti bet kokį vykdytoją, kuris atlieka ar apdoroja paskirtąją užduotį. Tokiu procesoriumi gali būti asmuo, agregatas ar bet koks kitas įtaisas, apdorojantis paskirtas užduotis. Atskiri procesoriai, vykdantys tą pačią užduotį, gali skirtis užduočių vykdymo intensyvumu (tempu, greičiu).

Šiame darbe nagrinėsime uždavinius, kuriuose visi procesoriai atlieka užduotis vienodu intensyvumu. Pastaruoju atveju galima įvesti atskirą išteklių tipą, žymintį procesorius, reikalingus užduotims atlikti. Tad užduočiai atlikti reikalingos vykdymo išteklių rūšys yra laikomos procesorių grupėmis, kuriose yra po tam tikrą skaičių procesorių. Viena užduotis yra išskaidoma į atskiras užduotis toms procesorių grupėms ir kiekvienai iš tų išskaidytų užduočių atlikti reikalingas tam tikras skaičius procesorių. Laikoma, jog visų grupių procesoriai išskaidytas užduotis atlieka per tą patį nustatytą laiką, lygų duotai užduoties trukmei. Jei tuo metu atskirose procesorių grupėse esančių laisvų procesorių pakanka, kad atlikti kitą išskaidytą užduotį, tada šią užduotį galima vykdyti lygiagrečiai. Jei nors vieno procesoriaus iš bet kurios jų grupės nepakanka, užduotis negali būti vykdoma. Pavyzdžiui, standartinėje testinių uždavinių bazėje PSPLib (žiūr. 7.1 skyrelį) ištekliai interpretuojami kaip 4 procesorių grupės, kurių kiekvienoje yra tam tikras lygiaverčių

procesorių skaičius. Bet kuriuo projekto vydymo momentu atliekant kai kurias užduotis lygiagrečiai (jei nenusižengiama nuoseklumo sąryšiams) atliekamų užduočių procesorių poreikiai negali viršyti turimų jų ribų. Jei laisvų procesorių užduočiai atlikti nepakanka, užduoties vykdymą tenka atidėti, kol vėl bus laisvas reikiamas kiekis procesorių, reikalingas užduotims atlikti.

3.3 pavyzdys. Tarkime 3.3 pav. duomenyse užduotims atlikti galima panaudoti 4 skirtingų tipų procesorių grupes, pirmąjai užduočiai atlikti reikia dviejų pirmosios ir dviejų ketvirtosios grupės procesorių, o antrąjai reikia po tris antros ir trečios grupės procesorių ir t.t. Kiekvienoje grupėje yra po keturis procesorius. Tad pirmąjąją ir antrąją užduotis galima pradėti vykdyti lygiagrečiai, nes vykdančių procesorių tam pakanka.

3.8 Tvarkaraščių optimizavimo kriterijai

Optimizuojant tvarkaraščius dažniausiai tenka rinktis vieną ar keletą optimizavimo kriterijų. Dažniausiai yra taikomi tokie kriterijai:

- minimalus viso projekto užbaigimo laikas (*Makespan*),
- minimalus viso projekto vėlavimas, nuo nustatyto užbaigimo laiko,
- kompleksiniai kriterijai,
- minimalios sąnaudos, išlaidos,
- maksimalus pelnas,
- optimalus vykdytojų (procesorių) panaudojimas.

Daugelyje planavimo, statybos ir kitų uždavinių vienu iš svarbiausių optimizavimo kriterijų yra viso projekto užbaigimo laikas, kuris paprastai yra minimizuojamas. Pramonėje sutinkamas kitas optimizavimo kriterijus – maksimalus vėlavimas, kuris taip pat minimizuojamas. Iš kelių kriterijų galima sukonstruoti ir kompleksinius svorinius optimizavimo kriterijus. Pavyzdžiui, visas svorinis baigimo laikas (total weighted completion time, $\Sigma\ w_jC_j$), bendras svorinis vėlyvumas (total weighted tardiness, $\Sigma\ w_jT_j$), svorinis skaičius pavėluotų užduočių ($\Sigma\ w_jU_j$). Priklausomai nuo uždavinių specifikos, taip pat gali būti minimizuojamos sąnaudos (išlaidos), maksimizuojamas pelnas, optimizuojamas įrenginių, vykdytojų, procesorių skaičius.

3.9 Pirmumo sąrašas, sprendinių vaizdavimas bei gretimų sprendinių gavimas

3.9.1 Topologiniai tvarkaraščių formalizavimo metodai

Sudarant tvarkaraštį reikia išnagrinėti įvairius tvarkaraščio variantus, pereinant nuo vieno tvarkaraščio prie kito. Topologinių metodų taikymas, įvedant topologiją planuojamų užduočių aibėje, palengvina tokių perėjimo metodų formalizavimą. Vienas iš būdų įvesti topologiją yra nagrinėjamos aibės metrizavimas, nagrinėjant ją kaip metrinės erdvės poaibį (B priedas). Metrikai ir topologijai tvarkaraščių užduočių aibėje įvesti naudojami du būdai. Jeigu tvarkaraštį koduotume

užduočių pradžių ir pabaigų vektoriais, tada tvarkaraštį galime nagrinėti kaip euklidinės erdvės elementą. Šiuo atveju tvarkaraštį atitinka užduočių pradžių (ar pabaigų) vektorius x_0 daugiamatėje erdvėje. Rutulį $B(x_0,r)$ tokiu atveju sudaro tvarkaraščiai, kurių užduočių pradžių (pabaigų) vektorių atstumas nuo x_0 neviršyja r euklidinėje metrikoje. Remiantis tokiu formalizavimu, galima formuluoti tvarkaraščio optimizavimo uždavinį, tačiau tokio uždavinio sprendimas bendru atveju yra sudėtinga globaliojo optimizavimo problema. Todėl reikia ieškoti kitų tvarkaraščio formalizavimo būdų.

Tvarkaraščiams su ribojimais spręsti dažnai yra taikomas kodavimas užduočių pirmumo leistino sąrašo pavidalu. Leistinu pirmumo sąrašu vadinsime tokį, kuriam galima rasti užduočių pradžių laikų vektorių, suderintą su nuoseklumo sąryšiais. Remiantis nuoseklumo sąryšių aibe, nesunku nustatyti ar sprendinys yra leistinas, ar ne. Bet kuriam leistinam užduočių pirmumo sąrašui, pritaikius nuoseklaus dekodavimo procedūrą galime nustatyti užduočių pradžių vektorių, minimizuojantį projekto atlikimo laiką, atsižvelgiant į nuoseklumo sąryšius, išteklių ribojimus bei įvestą užduočių pirmumo sąrašą. Gretimais pirmumo sąrašais laikomi tie, kurie gaunami vienas iš kito, atlikus elementarias operacijas. Elementariomis operacijomis dažniausiai yra laikomos užduočių sukeitimo arba perkėlimo operacijos. Šiuo atveju topologizuojama aibė X yra visų pirmumo sąrašų perstatinių aibė. Atstumas tarp dviejų pirmumo sąrašų yra lygus minimaliam elementarių operacijų skaičiui, kuris reikalingas, kad iš vieno pirmumo sąrašo gauti kitą. Kiekvienam pirmumo sąrašui ir duotam r galima nustatyti spindulio r rutulį (r gylio aplinką), kurį sudaro visi tvarkaraščiai, gaunami iš pradinio atlikus ne daugiau nei r elementarių operacijų. Topologiją sudaro visų taškų, visų galimų aplinkų junginys.

Reikia pažymėti, kad tvarkaraščių uždavinio topologiją galima formalizuoti nebūtinai vieninteliu būdu. Pavyzdžiui, lygiagrečių užduočių tvarkaraščiams (be nuoseklumo sąryšių) galima taikyti topologizavimo būdą, skirtingą nuo pirmumo sąrašo. Jeigu užduotis atlieka tam tikras procesorių skaičius, kiekvienai užduočiai reikia priskirti procesorių, kuris vykdys šią užduotį. Tuomet tvarkaraštį apibūdins užduočių eilei priskirta jas vykdančių procesorių numerių seka. Šiuo atveju gretimais tvarkaraščiais bus galima laikyti tuos, kuriuose procesorių numerių seka skiriasi tik vienu numeriu. Atstumas tarp dviejų tvarkaraščių yra matuojamas procesorių sekos skirtingų numerių skaičiumi. Aplinką arba rutulį $B(x_0,r)$ sudaro visi tvarkaraščiai, kurie atliekami ne daugiau negu r skirtingų procesorių negu tvarkaraštis x_0.

3.9.2 Pirmumo sąrašas

Pirmumo sąrašu yra vadinamas (n+2) komponenčių vektorius $b=(0,b_1,b_2,...,b_n,b_{n+1})$, kurį sudaro užduočių numeriai (kartu su fiktyviomis pradžios ir pabaigos užduotimis), pavyzdžiui, b = (0,1,2,3,4,5). Šio sąrašo komponentę atitinkanti užduotis negali prasidėti anksčiau, negu užduotys, atitinkančios komponentes, einančias prieš ją. Darbų pradžios gali būti nustatytos

naudojant dekodavimo algoritmą (žiūr. 6.3 skyrelį). Tvarkaraščiai taip pat gali būti koduojami naudojant darbų pabaigų dekoderį ir lygiagretųjį dekoderį (Kolisch, 1996b, Kocetov, Stoliar, 2003).

Gretimi sprendiniai gaunami perstatinėjant pirmumo sąrašo komponentes. Perstatymai atliekami taip, kad būtų galima gauti bet kokią įmanomą komponenčių kombinaciją.

Įvedame užduoties (užduočių) perkėlimo į kitą poziciją operacijas. *Pirmoji – užduoties perkėlimo operacija* (3.10 pav.)*, antroji – užduočių sukeitimo operacija* (3.11 pav.). Nors kiekviena iš šių operacijų gali būti atlikta, keletą kartų pritaikius kitą, paprastumo dėlei algoritmuose yra naudojamos abi šios operacijos.

$\langle\, 0,1,2,\mathbf{3},4,5 \,\rangle \Rightarrow \langle\, 0,\mathbf{3},1,2,4,5 \,\rangle$

3.10 pav. Perkėlimo (shift) operacija

$\langle\, 0,\mathbf{1},2,\mathbf{3},4,5 \,\rangle \Rightarrow \langle\, 0,\mathbf{3},2,\mathbf{1},4,5 \,\rangle$

3.11 pav. Sukeitimo (swap) operacija

Perkeliant ar sukeičiant užduotis yra labai svarbu nepažeisti nuoseklumo sąryšių. Prieštaravimų nuoseklumo sąryšiuose galima išvengti įvairiais būdais. Pateikiame du metodus, kurie padeda išvengti nuoseklumo sąryšių pažeidimų.

I. Patikrink-perkelk (Check-Move):

1. Patikriname, ar keliama užduotis nepažeidžia nuoseklumo sąryšių.

2. Jei nepažeidžia – perkeliame, priešingu atveju – neperkeliame.

Uždraudus perkėlimą yra ieškoma kito gretimo tvarkaraščio. Pavyzdžiui, $\langle\, 0,1,2,\mathbf{3},4,5 \,\rangle \Rightarrow \langle\, 0,\mathbf{3},1,2,4,5 \,\rangle$. Tarkim, jei egzistuoja nuoseklumo sąryšiai (1,2) ir (2,3), tai galime įsitikinti, kad tokia planuojama perkėlimo operacija nebus vykdoma, nes 3-ioji užduotis negali būti vykdoma neatlikus 1-osios ir 2-osios. Uždraudus tokį perkėlimą, bus ieškoma kito gretimo tvarkaraščio, kurį galima gauti perkeliant užduotis.

II. Perkelk-sutvarkyk (Move-Repair):

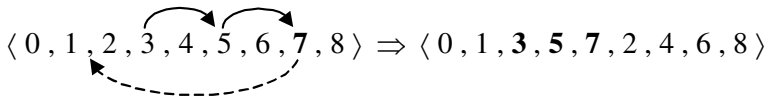
1. Perkeliame numatytą užduotį į kitą poziciją.

2. Jei pažeisti nuoseklumo sąryšiai, tvarkome kilnodami ir kitas užduotis.

Pavyzdžiui, $\langle\, 0,1,2,\mathbf{3},4,5 \,\rangle \Rightarrow \langle\, 0,1,\mathbf{3},2,4,5 \,\rangle$. 3-ioji užduotis perkelta prieš 2-ąją. Kadangi gautas naujas vektorius prieštarauja nuoseklumo sąryšiui (2,3), tai su perkelta užduotimi susijusios užduotys irgi perkeliamos į priekį, atkuriant nuoseklumo sąryšius:

$\langle\, 0,1,3,\mathbf{2},4,5 \,\rangle \Rightarrow \langle\, 0,\mathbf{2},1,3,4,5 \,\rangle$ arba $\langle\, 0,1,3,\mathbf{2},4,5 \,\rangle \Rightarrow \langle\, 0,1,\mathbf{2},3,4,5 \,\rangle$ (šiuo atveju vėl gautume pradinį sprendinį).

Nuoseklumo sąryšių atkūrimą iliustruosime kitu pavyzdžiu. Tarkime, turime vektorių užduočių numerių. Keletas užduočių yra tarpusavyje susijusios nuoseklumo sąryšiais (3.12 pav.).



3.12 pav. Užduoties perkėlimas ir nuoseklumo sąryšių atstatymas

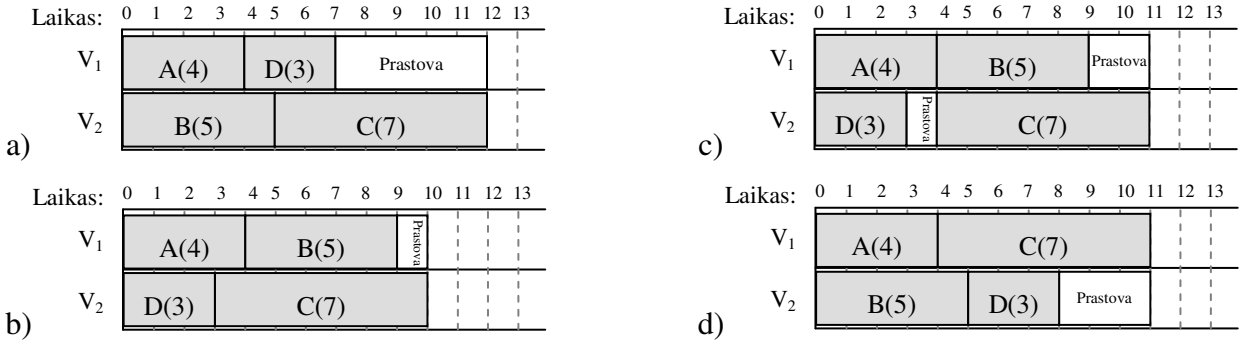
Nuoseklumo sąryšiai (3,5) ir (5,7) pavaizduoti virš jų nubrėžtomis rodyklėmis. Perkėlus (punktyrinė rodyklė) 7-ąją užduotį į priekį, nusižengiama šiems nuoseklumo sąryšiams. Sąryšiais susijusios užduotys taip pat perkeliamos į priekį, atkuriant buvusius sąryšius. Tokiu būdu, visos trys užduotys – 3, 5, 7 - atsiduria iškart po užduoties 1.

Panagrinėsime tvarkaraščio su ribojimais pavyzdį, kuris iliustruoja pirmumo sąrašo įtaką viso projekto vykdymo laikui.

3.4 pavyzdys. Turime du procesorius V_1 ir V_2, kuriems reikia sudaryti tvarkaraštį tokioms keturioms užduotims atlikti (vykdymo trukmės valandomis nurodytos skliausteliuose): *A(4), B(5), C(7), D(3).* Vienintelis nuoseklumo sąryšis yra $A \rightarrow C$.

3.13 a) pav. parodytas tvarkaraštis, kuris yra leistinas, bet neefektyvus. Visos trumpalaikės užduotys buvo pavestos vienam procesoriui (V_1), o visos ilgalaikės – kitam procesoriui (V_2).

3.13 b) pav. parodytas geresnis tvarkaraštis, tačiau jis pažeidžia nuoseklumo sąryšį $A \rightarrow C$.



3.13 pav. Keletas galimų 3.4 pavyzdžio tvarkaraščių

3.13 c) pav. tvarkaraščio vykdymo trukmė yra 11 valandų. Kadangi užduoties *C(7)* negalima pradėti vykdyti anksčiau kaip po keturių valandų – tai yra trumpiausias įmanomas tvarkaraštis. *Joks tvarkaraštis neleidžia baigti šio projekto greičiau kaip per* 11 *valandų* (Tannenbaumas, Arnoldas, 1995).

Bendru atveju gali būti daugiau kaip vienas optimalus tvarkaraštis. 3.13 d) pav. parodytas kitas tvarkaraštis, kurio vykdymo trukmė taip pat yra 11 valandų.

Tvarkaraščių euristinės paieškos algoritmai, besiremiantys kodavimu darbų pirmumo sąrašu, išsamiau yra nagrinėjami 6-ajame skyriuje.

3.10 Išvados

1. Tvarkaraščio formalus aprašymas sudaromas naudojant pradinius duomenis apie užduočių trukmių laikus, nuoseklumo saryšius, išteklius, reikalingus užduotims atlikti bei išteklių ribojimus.
2. Tvarkaraštis yra laikomas sudarytu, jei yra nustatyti užduočiu pradžios ar pabaigos laikai. Šie laikai yra randami naudojantis pradiniais duomenimis, sprendžiant optimizavimo uždavinį su pasirinktu kriterijumi (pvz. minimaliu viso projekto atlikimo laiku, minimaliomis sąnaudomis ir pan.).

4 skyrius. Perrinkimo ir leistinųjų sprendinių radimo metodai

Šiame skyriuje panagrinėsime tipinius sveikaskaičio programavimo uždavinius, susijusius su RITSU bei jų sprendimo klasikinius metodus. Tokiems uždaviniams spręsti yra taikomi perrinkimo, sveikaskaičio tiesinio programavimo metodai (šakų ir rėžių metodas, Gomori algoritmai) ir kiti. Kadangi didžioji dauguma tvarkaraščių uždavinių sprendimo algoritmų yra NP sudėtingumo, optimalius tokių uždavinių sprendinius per priimtiną laiką galime rasti tik mažos apimties uždaviniams. Praktinėse situacijose dažniausiai tenka tenkintis apytikriu sprendiniu, siekiant, kad jis būtų kuo artimesnis optimaliam. Tokius sprendinius galima gauti naudojant daugelį klasikinių *sveikaskaičio programavimo algoritmų*. Šie sprendiniai gali būti toliau gerinami euristinės paieškos metodais, nagrinėjamais 5-ajame ir 6-ajame skyriuose.

4.1 Sveikaskaičiai optimizavimo uždaviniai, susiję su tvarkraščių planavimu

4.1.1 Keliaujančio pirklio uždavinys

Tai klasikinis grafų teorijos (žiūr. A priedą) uždavinys, su kuriuo susiduriama daugelyje įvairių transporto organizavimo, tvarkaraščių sudarymo problemų. Šio uždavinio pagrindinis tikslas yra rasti optimalų (trumpiausią, mažiausiai sąnaudų reikalaujantį ir pan.) maršrutą svoriniame grafe, pradedant kelionę iš vienos viršūnės, aplankant visas kitas tik po vieną kartą bei grįžtant į pradinę viršūnę. Tokį minimizavimo uždavinį tektų spręsti keliautojui, planuojančiam savo kelionę ir besiruošiančiam aplankyti keletą žymių vietų bei grįžti atgal. Buitinių atliekų surinkimo uždavinys mieste ar autobuso maršruto planavimo uždavinys, apvažiuojant tam tikras stoteles, taip pat gali būti interpretuojami, kaip keliaujančio pirklio uždaviniai.

Matematiškai šis uždavinys formuluojamas taip. Sakykime, yra n viršūnių (miestų, sandėlių, jūrų uostų ar kitų objektų), kurias žymėsime $i = 1,...,n$. Atstumai c_{ij} tarp bet kurių *i* ir *j* punktų yra žinomi ir aprašomi matrica $c = \lfloor c_{ij} \rfloor,\ i, j = 1,...,n$. Pradėjus nuo viršūnės 1, reikia apeiti visas viršūnes tik po vieną kartą trumpiausiu keliu, ir grįžti į pradinę viršūnę. Taigi, reikia nustatyti grafo trumpiausią paprastą ciklą iš *n* briaunų, kai grafo su *n* viršūnių briaunų *ij* ilgiai c_{ij} yra žinomi. Toks uždavinys tikrai turi sprendinį: yra (*n*-1)!/2 skirtingų ciklų (du priešingų krypčių ciklai laikomi vienodais), iš kurių vienas (arba keli vienodo ilgio) yra trumpiausias. Pažymėkime nežinomuosius:

$$x_{ij} = \begin{cases} 1, & jeigu\ komivojažierius\ iš\ punkto\ i\ vyksta\ į\ punktą\ j, i \neq j, \\ 0, & kitais\ atvejais. \end{cases}$$

Keliaujančio pirklio uždavinys formuluojamas taip: rasti

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}\ , \tag{4.1}$$

kai $\sum_{i=1}^{n} x_{ij} = 1$, $j = 1,...,n$, $i \neq j$; $\sum_{j=1}^{n} x_{ij} = 1$, $i = 1,...,n$, $i \neq j$;

$u_i - u_j + nx_{ij} \leq n-1$, $i, j = 2,...,n$, $i \neq j$;

$x_{ij} = 0$ arba $x_{ij} = 1$, $i, j = 1,...,n$.

Papildomos sąlygos $\sum_{i=1}^{n} x_{ij} = 1$ ir $\sum_{j=1}^{n} x_{ij} = 1$ reiškia ribojimą, jog kiekviena viršūnė būtų aplankyta tik vieną kartą. Tačiau šių ribojimų nepakanka paprastam ciklui, einančiam per visas viršūnes, gauti. Viršūnių apėjimas gali išsiskirti į keletą tarpusavyje nesusietų ciklų, einančių per mažesnį už n viršūnių skaičių. Tad sąlygos $u_i - u_j + nx_{ij} \leq n-1$, $i, j = 2,...,n$, $i \neq j$ yra įvedamos tam, kad išvengti tokio atvejo ir užtikrinti, jog apėjimo maršrutas būtų vientisas.

Šį uždavinį galima formuluoti, įvedus viršūnių perėjimo pirmumo sąrašą b=(1, b_2, ..., b_n), kurį sudaro grafo viršūnių numeriai b_i, surašyti jų apėjimo tvarka. Tada turėsime kombinatorinio optimizavimo uždavinį

$$\min_{b \in \Im_{n-1}} \left(c_{1,b_n} + \sum_{i=2}^{n} c_{b_{i-1} b_i} \right), \tag{4.2}$$

čia $\Im_{n-1}$ – visų (n-1)-tos eilės kėlinių aibė.

Šį uždavinį galima interpretuoti kaip gatavos produkcijos tiekimo vartotojams, medžiagų tiekimo įvairiems statybos objektams, buitinių atliekų surinkimo optimalaus maršruto nustatymo uždavinį, pigiausios kelionės, aplankant numatytas vietas, nustatymo uždavinį, ir panašiai.

4.1.2 Kuprinės uždavinys

Šie uždaviniai atspindi labai daug šiuolaikinių realių uždavinių. Yra nemažai šio uždavinio modifikacijų ir variacijų. Visų šių uždavinių esmė – suskirstyti tam tikrų objektų aibę į atskirus poaibius, atsižvelgiant į tam tikrus reikalavimus, kurie skirtingoms uždavinių variacijoms gali būti skirtingi. Pavyzdžiui, duoti daiktai su žinomomis vertėmis ir apimtimis (tūriais, svoriais). Rasti daiktų rinkinį, kurių vertė būtų didžiausia ir kurie tilptų į duoto fiksuoto dydžio kuprinę. Kiekvieno tipo daiktų skaičius, kurį galima paimti, gali būti ribojamas (tokiu atveju nurodomas mažiausias ir/arba didžiausias leidžiamas paimti objektų kiekis) arba neribojamas.

Yra žinomos kelios šio uždavinio modifikacijos. Pradėkime nuo paprasčiausių. Tarkime, turime n nedalomų skirtingų objektų (daiktų), kurių vertė ir svoris yra atitinkamai c_j ir a_j, $j = 1,...,n$. Į kuprinę galima sudėti ne daugiau kaip b svorio vienetų krovinį. Atsižvelgiant į kuprinės talpą (ar maksimalų leistiną svorį), reikia nustatyti kraunamų į kuprinę maksimalios vertės daiktų rinkinį. Įveskime kintamuosius x_j, kurių prasmė tokia: $x_j = \begin{cases} 1, & \text{jeigu } j-\text{ąjį daiktą dedame į kuprinę,} \\ 0, & \text{jeigu } j-\text{ojo daikto į kuprinę nededame.} \end{cases}$

Nagrinėjamas uždavinys išreiškiamas šiuo matematiniu modeliu: reikia rasti

$$max \sum_{j=1}^{n} c_j x_j , \tag{4.3}$$

kai $\sum_{j=1}^{n} a_j x_j \le b$, $x_j = 0$ arba $x_j = 1$, $j = 1,...,n$.

Kadangi kintamųjų x_j reikšmės yra 0 arba 1, tai jie vadinami binariniais. Šis uždavinys dar vadinamas binariniu kuprinės uždaviniu. Kuprinės sąvoka šiuose uždaviniuose yra apibendrinta. Už jos gali slypėti įvairūs objektai: transporto priemonė, sandėlis ar bet kokia kita talpykla, vietoje svorio apribojimo gali būti bendro tūrio apribojimas ir panašiai.

Panagrinėkime šio uždavinio pritaikymą investicijų paskirstymo uždavinyje, esant tam tikriems ribojimams. Tarkime, turime *b* apimties kapitalą, kurį galima panaudoti n būdų (projektų), $j = 1,...,n$. Yra žinomos išlaidos a_j, susijusios su kiekvienu projektu *j*, bei atitinkamas pelnas c_j, gaunamas realizavus projektą. Tegul visa investuojama suma *b* yra mažesnė už išlaidas, reikalingas visiems projektams įvykdyti, projektai nepriklauso vienas nuo kito. Tad išlaidos ir pelnas gaunami sumuojant pasirinktų projektų dydžius a_j ir c_j. Reikia parinkti optimalų projektų rinkinį, garantuojantį didžiausią pelną. Todėl pasirinkus binarinius kintamuosius $x_j = 0$ arba $x_j = 1$, priklausomai nuo to, ar pasirenkamas *j*-asis projektas ar ne, gaunamas jau aprašytas modelis.

Yra žinoma daug kuprinės uždavinio dvimačių, trimačių ir kitokių apibendrinimų, optimizuojant įvairius kriterijus. Daugiamatis kuprinės uždavinys yra pritaikomas sudarant algoritmus optimaliems tvarkaraščiams rasti kintamos aplinkos metodu (Kocetov, Stoliar, 2003).

4.1.3 Pakavimo uždavinys

Su kuprinės uždaviniu artimai susijęs yra pakavimo uždavinys (*binpacking*). Šiame uždavinyje reikalaujama nustatyti, kaip reikia sudėti didžiausią kiekį objektų į mažiausią skaičių fiksuotos apimties talpyklų (dėžių, konteinerių). Šis uždavinys dažnai sutinkamas gamyboje. Dviejų talpyklų atveju, tam tikrą skaičių aibę reikia suskirstyti į dvi dalis taip, kad skirtumas tarp abiejų aibių elementų sumų būtų mažiausias. Šį uždavinį galima interpretuoti kaip užduočių paskirstymo dviems procesoriams tvarkaraščio uždavinį be nuoseklumo sąryšių.

Pakavimo uždavinys formuluojamas taip:

$$F(x) = \sum_{i=1}^{n} \left| x_i y_i \right| \to min, \; y_i = \{-1,1\}, \; x_i \in R. \tag{4.4}$$

Pvz., turime du suskirstymus S1 ir S2 (y_i reikšmės).

S1: 1, 1, -1, 1, -1, -1, 1, -1, 1, 1

S1: 1, -1, 1, 1, 1, -1, -1, 1, -1, 1

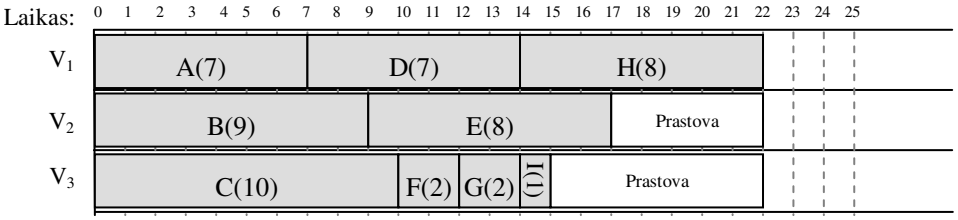
Kai objektų (užduočių) nedaug, galima perrinkti visus įmanomus variantus ir išrinkti geriausią suskirstymą. 4.2 skyrelyje pateikiami kompiuterinio modeliavimo rezultatai, parodantys, kad

perrinkimo metodu galima spręsti uždavinius, kai užduočių skaičius yra 35 ÷ 40. Kai užduočių yra daugiau, tenka ieškoti kitų metodų optimaliam arba artimam optimaliam sprendiniui rasti. Skyrelyje 4.3.1. nagrinėjamas šio uždavinio sprendimas mažėjančių trukmių algoritmu.

4.1.4 Nepriklausomų užduočių tvarkaraščių uždaviniai

Su pakavimo uždaviniais yra susiję nepriklausomų užduočių tvarkaraščių uždaviniai. Tegul turime užduočių, kurias reikia priskirti keliems procesoriams, aibę, kai žinomos užduočių vykdymo trukmės, ir nėra ribojimų užduočių eiliškumui. Tokių užduočių priskyrimo keliems procesoriams uždaviniai yra nepriklausomų užduočių tvarkaraščių uždaviniai. Bendru atveju šis uždavinys yra NP sudėtingumo ir nėra žinoma efektyvių algoritmų optimaliam sprendiniui rasti. Nors nepriklausomų užduočių tvarkaraščius sudaryti yra ne ką paprasčiau, negu esant nuoseklumo sąryšiams, tačiau visa procedūra, naudojama tvarkaraščiui sudaryti pagal pirmumo sąrašą, labai supaprastėja.

4.1 pavyzdys. Tarkime, kad turime devynias nepriklausomas užduotis: *A(7), B(9), C(10), D(7), E(8), F(2), G(2), H(8) ir I(1).* Reikia sudaryti šių užduočių tvarkaraštį trims vienodo pajėgumo procesoriams (V_1, V_2 ir V_3). Pirmiausiai tai padarykime, naudodamiesi atsitiktiniu pirmumo sąrašu: *A(7), B(9), C(10), D(7), E(8), F(2), G(2), H(8) ir I(1).* Šiuo atveju tvarkaraštis parodytas 4.1 pav. Vykdymo trukmė – 22 valandos.

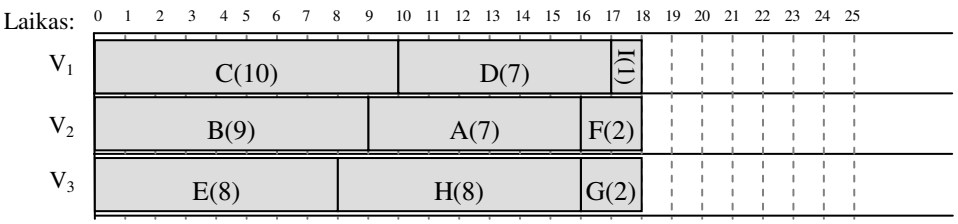


4.1 pav. Tvarkaraštis pagal 4.1 pavyzdžio duomenis su atsitiktiniu pirmumo sąrašu

Jei naudosimės kritinių kelių algoritmu, tai pirmumo sąrašas bus mažėjančių trukmių sąrašas:

• **Mažėjančių trukmių sąrašas:** *C(10), B(9), E(8), H(8), A(7), D(7), F(2), G(2), I(1).*

Tvarkaraštis yra optimalus (visi trys procesoriai dirba po 18 valandų, žiūr. 4.2 pav.).



4.2 pav. Tvarkaraštis su mažėjančių trukmių pirmumo sąrašu

4.1 pavyzdyje kritinių kelių algoritmu gavome optimalų tvarkaraštį, tačiau tai nėra taisyklė. Kai kuriose situacijose kritinių kelių algoritmu galima gauti prastokus tvarkaraščius.

Mes galime išmatuoti kritinių kelių algoritmo „gerumą“ santykinės procentinės paklaidos terminais.

Pvz.: (22-18)/18=2/9=22,22%

Tam tikra prasme 4.1 pavyzdžio situacija yra pati nepalankiausia kritinio kelio algoritmui.

Amerikiečių matematikas Graham (Graham, 1969) įrodė, kad kritinių kelių algoritmu nepriklausomoms užduotims visada gausime tvarkaraščius, kurių vykdymo trukmė gali skirtis nuo optimalios tik tam tikru palyginti nedideliu procentu, ir nustatė, kaip didžiausia procentinė paklaida priklauso nuo procesorių skaičiaus (4.1 lentelė).

4.1 lentelė. Kritinių kelių algoritmo didžiausia galima procentinė paklaida, kai užduotys nepriklausomos

Procesorių skaičius	Didžiausia paklaida
2	*16,16%*
3	*22,22%*
4	*25,00%*
5	*26,66%*
...	...
N	*(N-1)/(3N)*

Nors per daugelį pastarųjų metų atrasta daug kitų strategijų tvarkaraščiams sudaryti, tačiau iki šiol nėra žinomas toks algoritmas, kuris būtų efektyvus ir kuriuo būtų galima gauti optimalų tvarkaraštį. Gali būti, jog toks algoritmas iš viso neįmanomas.

4.1.5 Apibendrintas $P|\#\leq k|C_{max}$ uždavinys

Su šiuo uždaviniu susiduriama įvairiuose gamybos bei surinkimo procesuose, kai reikia vykdytojams paskirstyti įvairias operacijas (žiūr. 7.6 skyrelį, HP montažinių plokščių surinkimo uždavinys). Sprendžiant tokius taikomuosius uždavinius susiduriama su kombinatoriniu uždaviniu, kuris literatūroje apibrėžiamas kaip $P|\#\leq k|C_{max}$. Tai užduočių paskirstymo keliems lygiagretiems procesoriams uždavinys. Šis uždavinys spręstas įvairiais metodais (Dell'Amico, Iori, Martello, 2004).

Detalizuokime šį uždavinį. Yra duota n užduočių, kurių kiekvienos apdorojimo trukmė yra p_j ($j = 1, ..., n$), ir m identiškų lygiagrečių procesorių, kurių kiekvienas bet kokiu momentu gali apdoroti daugiausiai vieną užduotį. Reikia paskirstyti visas užduotis procesoriams taip, kad visų užduočių atlikimo laikas (makespan) būtų minimalus. Šis uždavinys žymimas $P||C_{max}$ ir jis yra NP sudėtingumo klasės (Graham, Lawler, Lenstra, Rinnooy Kan, 1979).

Užduočių atlikimo trukmės – neneigiami skaičiai. Kad išvengti trivialių ar neišsprendžiamų atvejų, laikoma, jog $2 \leq m$, $2m \leq n$ ir $n \leq mk$. Galimi uždavinio $P|\#\leq k|C_{max}$ taikymai iškyla tada, kai

m procesorių turi atlikti *n* skirtingo tipo operacijų (pvz. robotai surinkimo linijoje). Realiose situacijose, kiekvienam procesoriui skirtingų operacijų, kurias jis gali atlikti, skaičius *k* gali būti ribojamas.

Uždavinio $P|\#\leq k|C_{max}$ sprendimas gali būti pritaikytas įvairiose robotizuotose surinkimo linijose. Hillier ir Brandeau tyrinėjo operacijų skirstymo uždavinį, taikant spausdintuvo montažinių plokščių surinkimo procese kompanijoje „Hewlett – Packard (HP)“ (Hillier, Brandeau, 1998).

4.1.6 Rūšiavimo uždaviniai

Tvarkaraščių sudarymo uždaviniuose dažnai tenka spręsti duomenų rūšiavimo uždavinius. Algoritmų analizėje duomenų rūšiavimo problema laikoma viena svarbiausių, kadangi ši operacija labai dažnai pasitaiko programavime, žiniatinklio naršymo ir paieškos programose. Efektyvaus rūšiavimo algoritmo pasirinkimas gali turėti lemiamą įtaką programos vykdymo spartai, didėjant duomenų kiekiui. Todėl svarbi rūšiavimo algoritmo charakteristika yra jo sudėtingumas – rūšiavimo spartos priklausomybė nuo duomenų apimties.

Duomenų rūšiavimo uždavinys apibrėžiamas taip:

Turint n elementų seką $(a_1, a_2, ..., a_n)$*, reikia išdėstyti šiuos elementus taip, kad gautume naują n elementų seką* $(a_1', a_2', ..., a_n')$*, tenkinančią salygą* $a_i <= a_j$*, kai* $i < j$.

Paprasčiausių rūšiavimo algoritmų (išrinkimo rūšiavimo algoritmas, įterpimo rūšiavimo algoritmas, burbulo metodas) sudėtingumas yra kvadratinis (žymima $O(n^2)$. Dažnai greičiausiu laikomo greitojo rūšiavimo algoritmo sudėtingumas daugeliu atveju yra $O(n \log n)$, tačiau rūšiuojant beveik surūšiuotus duomenis, šio algoritmo sudėtingumas siekia $O(n)$.

Algoritmo sudėtingumas svarbus tik esant dideliam duomenų kiekiui. 4.2 lentelėje palyginimui pateikta kiek skiriasi procesoriaus operacijų skaičius didėjant duomenų kiekiui, naudojant du skirtingus algoritmus – pirmasis naudoja 5 n^2 operacijų, o antrasis – 20 n lg n.

4.2 lentelė. Santykinis rūšiavimo algoritmų laikas

Duomenų elementų skaičius	Santykinis laikas burbulo algoritmu	Santykinis laikas greitojo rūšiavimo algoritmu
10	500	200
100	50 000	4 000
1000	5 000 000	60 000
10000	500 000 000	800 000

Jei duomenų kiekis nedidelis, mums dažniausiai visiškai nesvarbu, kiek mikrosekundžių bus vykdomas rūšiavimas, tačiau esant didesniems duomenų kiekiams šis skirtumas auga sparčiai (4.2 lentelė). Kita vertus, esant labai mažiems duomenų kiekiams geriau taikyti burbulo metodą, nes jis yra paprastesnis. Algoritmų sudėtingumas taip pat priklauso nuo duomenų savybių. Pavyzdžiui, burbulo metodas bus daug greitesnis, jei bus bandoma rūšiuoti surūšiuotus duomenis – tada jo

sudėtingumas yra *O(n)*. Dažniausiai įvertinamas algoritmų sudėtingumas vidutiniu atveju, dažnai blogiausiu atveju ir tik kartais – geriausiu. Tas pats algoritmas, būdamas labai greitas geriausiu atveju, gali būti labai blogas vidutiniu ar blogiausiu atveju. Naudojant daugiaprocesorinį kompiuterį ar paskirstytą kompiuterių tinklą galima pasiekti ir dar geresnių rezultatų. Geriausiu atveju pasiekiamas sudėtingumas $O((\log n)^2)$. 4.3 lentelėje pateikti dažniausiai sutinkami rūšiavimo algoritmai ir jų sudėtingumas (vertinant palyginimo operacijų atžvilgiu).

4.3 lentelė. Rūšiavimo algoritmų sudėtingumų lentelė

Algoritmas	Blogiausias	Vidutinis	Geriausias
Greitojo rūšiavimo (*quicksort*)	$O(n^2)$	$O(n \log n)$	$O(n \log n)$
Kombinuotas	$O(n \log n)$	$O(n (\log n)^2)$	
Krūvos (*heapsort*)	$O(n \log n)$	$O(n \log n)$	$O (n \log n)$
Sluoksnio (*Shell sort*)	$O(n^2)$	$O(n^{1,2})$	$O(n)$
Suliejimo (*mergesort*)	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Burbulo (*bubble*)	$O(n^2)$	$O(n^2)$	$O(n)$
Įterpimo (*insertion*)	$O(n^2)$	$O(n^2)$	$O(n)$
Išrinkimo (*selection*)	$O(n^2)$	$O(n^2)$	$O(n^2)$

4.2 Perrinkimo uždaviniai

Dauguma tvarkaraščių sudarymo uždavinių yra NP-pilnieji. Tad uždaviniui (3.1, 3.2) reikalingas laikas turi augti eksponentiniu greičiu $O(c^n)$ (žiūr. priedą C). Dažniausiai optimalų sprendinį galima rasti pilno perrinkimo būdu. Panagrinėkime pakavimo uždavinio, suvedamo į nepriklausomų užduočių tvarkaraščių uždavinį, sprendimą šiuo būdu, siekiant įvertinti tam reikalingas kompiuterio laiko sąnaudas. Tvarkaraščio uždavinio pagrindinę pradinę informaciją sudaro atliekamų užduočių trukmių vektorius. Taigi, pradinės informacijos kiekis yra nusakomas šio vektoriaus komponenčių skaičiumi, t.y. užduočių skaičiumi *n*.

Siekiant įvertinti esamos kompiuterinės įrangos galimybes, buvo atliktas kompiuterinis eksperimentas, sprendžiant perrinkimo būdu pakavimo uždavinius su skirtingu elementų skaičiumi. Atsitiktiniu būdu buvo generuojami skaičių rinkiniai $A=(a_1, a_2, ..., a_n)$. Šių rinkinių poaibiai buvo generuojami naudojant leksikografinį ir Grėjaus kodą (Savage, 1997). Pakavimo uždavinių sprendimo laikas, generuojant poaibius Grėjaus kodu dviem skirtingais kompiuteriais, esant skirtingam elementų skaičiui *n*, pateiktas 4.4 lentelėje. Skaičiavimai buvo atliekami tokiais kompiuteriais: kompiuteris1: AMD K6.2 333 MHz, 192 MB RAM PC133, kompiuteris2: AMD Athlon64 3000+ DDR 512 MB DDR PC400.

4.4 lentelė. Pakavimo uždavinio dviems procesoriams sprendimo laikas

Elementų skaičius	Kompiuteris1	Kompiuteris2
***n* = 20**	**4,47 sek.**	**0,70 sek.**
***n* = 30**	**~1 val 18 min.**	**~12 min.**
* *n* = 40	~54 d.	~9 d.
* *n* = 50	~152 m.	~24 m.

* – žymi atvejus, kai numatoma uždavinio sprendimo trukmė yra apskaičiuota teoriškai.

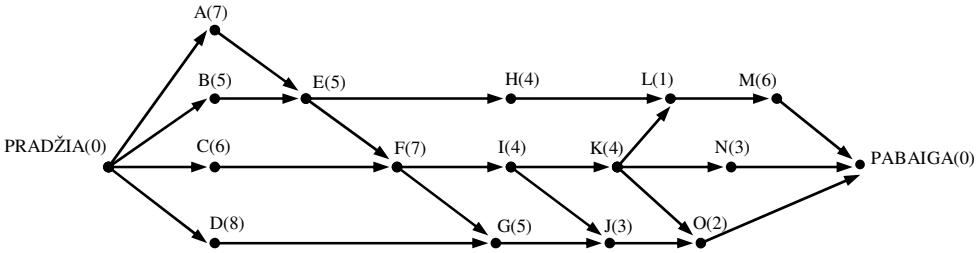
Taigi, pilną perrinkimą su turima kompiuterine įranga galima atlikti kai $n <= 30$, atskiru atveju $n = 35 \div 40$. Galingesnio kompiuterio naudojimas leidžia pagreitinti uždavinio sprendimą keletą ar keliolika kartų, tačiau užduočių skaičius, su kuriuo galima atlikti pilną perrinkimą, nuo to padidėja nedaug.

4.3 Tvarkaraščių uždavinių leistinųjų sprendinių radimo metodai

4.3.1 Mažėjančių trukmių algoritmas

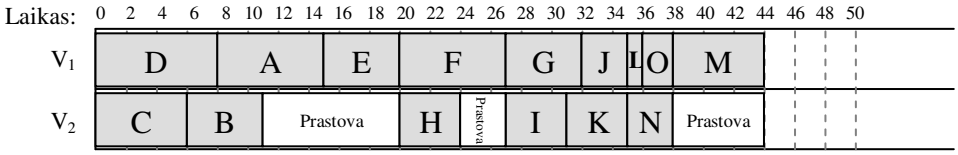
Panagrinėsime mažėjančių trukmių algoritmą, kuris taikomas leistiniems tvarkaraščių sprendiniams surasti. Šiame algoritme užduotys yra išdėstomos jų vykdymo trukmių mažėjimo tvarka. Tokiu būdu gautas sąrašas yra laikomas užduočių pirmumo sąrašu, pagal kurį yra nustatomi darbų pradžių vektoriai.

4.2 pavyzdys. Panagrinėkime pavyzdį, kurio duomenys pateikti 3.1-3.2 lentelėse bei 3.1 pav. Užduočių nuoseklumo orgrafas su papildytomis pradžios ir pabaigos užduotimis yra pateiktas 4.3 pav.



4.3 pav. Projekto duomenų atvaizdavimas orgrafu, pridėjus fiktyviąsias užduotis

Sudarykime tvarkaraštį naudojant tokį pradinį pirmumo sąrašą*: D(8), C(6), B(5), E(5), A(7), F(7), G(5), I(4), H(4), J(3), K(4), L(1), N(3), O(2), M(6).* (žiūr. 4.4 pav.)



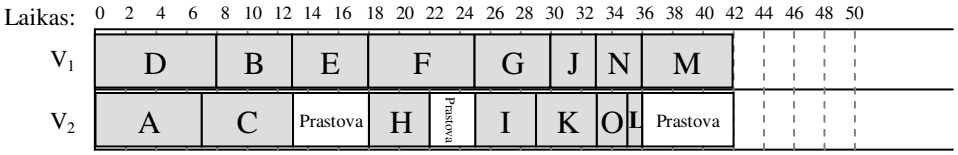
4.4 pav. Užduočių tvarkaraštis dviems vykdytojams

Gautas tvarkaraštis akivaizdžiai nėra optimalus. Naudojant kitokį pirmumo sąrašą būtų galima sprendinį pagerinti. Intuityviai atrodo racionalu ilgiau trunkančias užduotis atlikti anksčiau, negu trumpalaikes, t.y. taikyti mažėjančių trukmių algoritmą.

Pavyzdžiui, mažėjančių trukmių sąrašas 15-ai užduočių, atrodo taip:

D(8), A(7), F(7), C(6), M(6), B(5), E(5), G(5), I(4), H(4), K(4), J(3), N(3), O(2), L(1).

Remdamiesi šitokiu pirmumo sąrašu, gauname geresnį tvarkaraštį, parodytą 4.5 pav.



4.5 pav. Tvarkaraštis, sudarytas mažėjančių trukmių metodu

Tačiau šis sprendinys gali būti dar pagerintas. Matome, kad 33 valandą buvo trys galimos užduotys (*L(1), N(3)* ir *O(2))* ir du laisvi procesoriai. Mažėjančių trukmių algoritmas patarė mums rinktis dvi ilgiausias užduotis – *N(3)* ir *O(2).* Buvo neatsižvelgta į tai, kad reikia kiek įmanoma anksčiau pradėti užduotis *A(7)* ir *B(5).* Nebaigus jų, negalima pradėti *E(5),* o nebaigus *E,* negalima pradėti *F(7),* nebaigus *F,* negalime pradėti *I(4)* ir *G(5)* ir t.t. Iš čia galima spręsti, kad vertėtų teikti pirmumą užduotims, remiantis visų prieš ją einančių užduočių bendra vykdymo trukme.

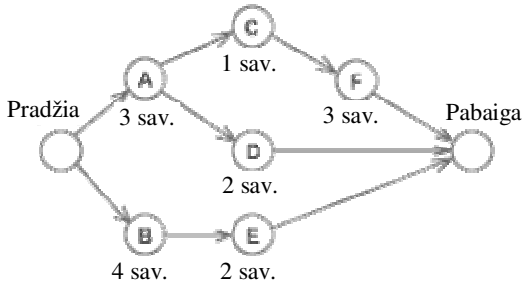
4.3.2 Kritinių kelių metodas

1957 metais buvo sukurtas projektų valdymo metodas, skirtas cheminių preparatų įrengimų patikrinimams juos išjungiant (Kelley, Walker, 1959, Kelley, 1961). Po patikrinimo įrengimai vėl buvo paleidžiami.

Kritinių kelių metodas naudingas dėl šių priežasčių:

- Suteikia galimybę gauti grafinį projekto vaizdą;
- Numato laiką, kurio reikia projektui baigti;
- Parodo, kurios iš užduočių yra kritinės tvarkaraščio sudarymui, o kurios ne.

Projekto užduotys ir įvykiai vaizduojami tinkline diagrama. Vienas iš dažniausiai naudojamų vaizdavimo būdų – „užduotys viršūnėse“ būdas, kuris ir buvo naudojamas kuriant šį metodą. 4.6 pav. pateiktas tokios tinklinės diagramos pavyzdys. Vėliau kiti tyrėjai naudojo ir kitą būdą – „užduotys briaunose“ vaizdavimą.



4.6 pav. Tinklinė diagrama

Žingsniai, kurie atliekami planuojant projektą kritinių kelių metodu:

1. Pagal darbo organizacinę struktūrą tiksliai nusakoma kiekviena užduotis. Taip sudaromas visų užduočių sąrašas ir duomenys apie užduotis (trukmės, nuoseklumas) gali būti papildyti vėlesniuose žingsniuose.
2. Nusakomas užduočių nuoseklumas. Pagal prieš kiekvieną užduotį būtinų atlikti užduočių sąrašą bus galima sudaryti tinklo diagramą.
3. Nubraižoma tinklo diagrama pagal užduočių nuoseklumo sąryšius.
4. Nustatoma kiekvienos užduoties vykdymo trukmė. Kritinių kelių metodas yra deterministinis metodas, kuriam nėra svarbūs trukmių svyravimai. Todėl užduoties trukmės nusakymui naudojamas vienas skaičius.
5. Nustatomas kritinis kelias (ilgiausios trukmės kelias tinklinėje diagramoje). Šis kelias svarbus tuo, jog užduotys esančios šiame kelyje negali būti vėlinamos be viso projekto vėlinimo. Kritinis kelias įtakoja visą projektą, todėl jį planuojant šio kelio analizė yra svarbi.

Kritinis kelias gali būti nustatomas kiekvienai užduočiai nusakant 4 parametrus:

- *ES* (earliest start) – ankstyvasis pradžios laikas, kada užduotis gali būti vykdoma, kai užbaigiamos prieš ją būtinos atlikti užduotys.
- *EF* (earliest finish) – ankstyvasis pabaigos laikas, kuris gaunamas prie ankstyvojo pradžios laiko pridėjus tos užduoties trukmę.
- *LF* (latest finish) – vėlyvasis pabaigos laikas, kada užduotis gali būti baigiama nevėlinant viso projekto.
- *LS* (latest start) – vėlyvasis pradžios laikas, kuris gaunamas iš vėlyvojo pabaigos laiko atėmus tos užduoties trukmę.

Laiko tarpas (slack time) tarp *ES* ir *LS* arba tarp *EF* ir *LF* yra galimas užduoties vėlinimas, kuris neturės įtakos viso projekto vėlinimui. *Kritinis kelias* yra kelias projekto tinkliniame grafe, kuriame nė vienai užduočiai nėra galimų vėlinimų, t.y. *ES=LS* ir *EF=LF* visoms užduotims šiame kelyje**.** Užduoties vėlinimas kritiniame kelyje vėlina visa projektą. Todėl tam, kad pagreitiniti viso projekto užbaigimą, yra būtina sutrumpinti laiką, kurio reikia atlikti užduotims, esančioms kritiniame kelyje.

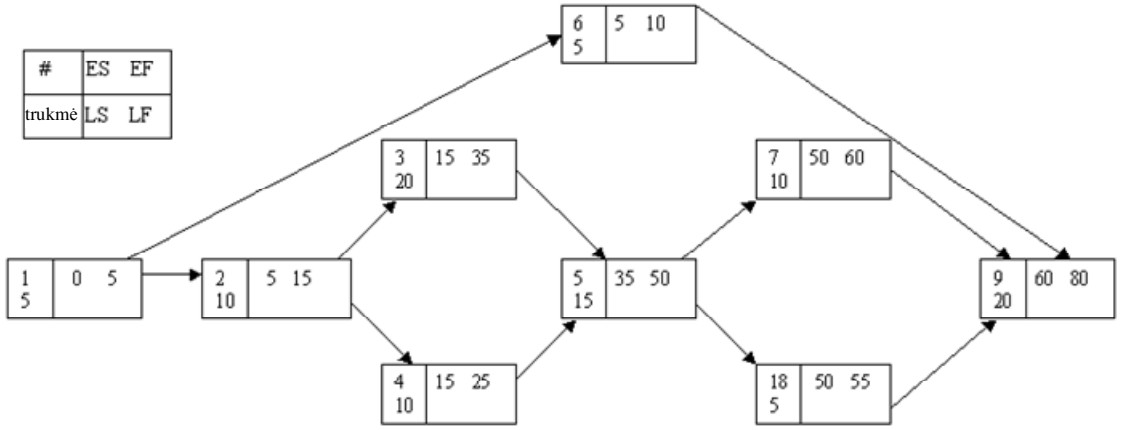
6. Projekto vykdymo metu atnaujinti tinklinę diagramą. Atliekant projekto užduotis, tikrieji baigimo laikai bus žinomi ir tinklo diagrama gali būti atnaujinama įtraukiant šią informaciją. Gali būti nustatomas naujas kritinis kelias ir gali būti padaromi struktūriniai pakeitimai tinkle, jei projekto sąlygos pasikeitė.

Projekto kritinis kelias nustatomas taip:

1. Visos užduotys surašomos į sąrašą, nurodomos jų trukmės p_i bei prieš jas būtinos atlikti užduotys (pagal nuoseklumo sąryšius) (žiūr. 4.5 lentelę.).
2. Nubraižomas užduočių nuoseklumą vaizduojantis orientuotasis grafas, kurio viršūnėse yra užduotį žymintys stačiakampiai (vienas iš patogiausių vaizdavimo būdų) su informacija apie ją, o grafo lankai žymi užduočių nuoseklumą.
3. Kiekvienai užduočiai nustatomi *ES* ir *EF*:
   - Užduotims *i*, prieš kurias nereikia atlikti kitų užduočių (tai gali būti ir fiktyvi užduotis, žyminti projekto pradžią), $ES(i) = 0$, o $EF(i) = ES(i) + p_i$.
   - Bet kuriai užduočiai *j*, prieš kurią turi būti atlikta aibė kitų užduočių (viena ar kelios), $ES(j) = max\{EF(i), (i,j) \in C\}$, $EF(j) = ES(j) + p_j$ (žiūr. 4.7 pav.).
4. Kiekvienai užduočiai nustatomi *LS* ir *LF*, peržiūrint diagramą iš kitos pusės:
   - Užduotims i, po kurių nereikia atlikti kitų užduočių (tai gali būti ir fiktyvi užduotis, žyminti projekto pabaigą), $LS(i) = ES(i)$, o $LF(i) = LS(i) + p_i$.
   - Bet kuriai užduočiai *i*, po kurios seka užduotis *j*, $LS(i) = ES(j) - p_i$, o $LF(i) = LS(i) + p_i$ (žiūr. 4.8 pav.).
5. Surandame *kritinį projekto kelią*. Užduotis *i* yra kritiniame kelyje, jei $ES(i) = LS(i)$ (žiūr. 4.9 pav.). Visos užduotys, kurios nėra kritiniame kelyje, gali būti vėlinamos (tam tikrame laiko intervale) neįtakojant viso projekto baigimo laiko (galimi ir keli skirtingi kritiniai keliai).
6. Projekto *kritinis laikas* (KL) lygus visų užduočių, esančių kritiniame kelyje, trukmių sumai.

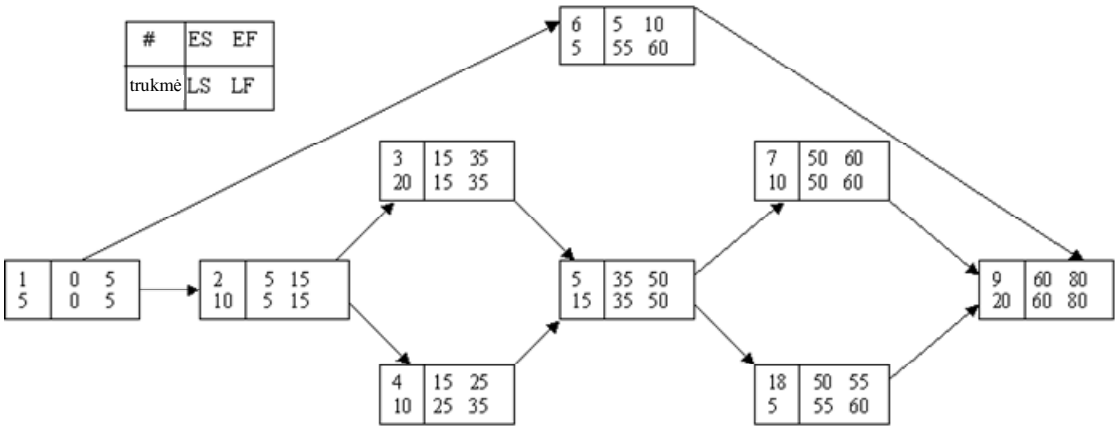
4.5 lentelė. Duomenys apie projekto užduotis

Užduoties nr.	trukmės	Pirmiau būtinos atlikti užduotys
1	5	-
2	10	1
3	20	2
4	10	2
5	15	3,4
6	5	1
7	10	5
8	5	5
9	20	6,7,8



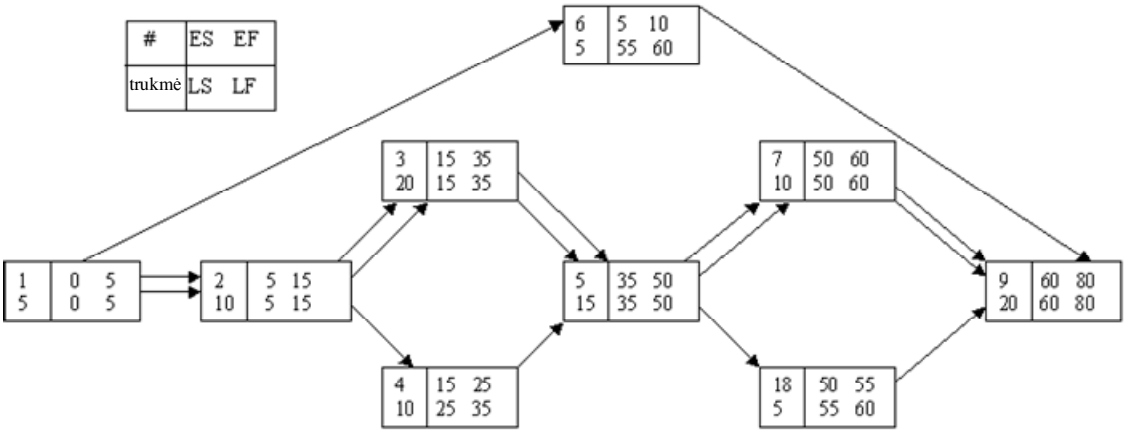
4.7 pav. Užduočių, jų trukmių, nuoseklumo sąryšių pateikimas grafu su užduočių *ES* ir *EF*

Pvz. 9-oji užduotis gali būti pradėta tik tada, kai pabaigtos 3 užduotys – 6-oji, 7-oji, 8-oji. Todėl jos ankstyvasis pradžios laikas bus lygus didžiausiam iš prieš ją būtinų atlikti užduočių ankstyvųjų baigimo laikų, t.y. $ES(9) = max\{10,60,55\} = 60$. Ankstyvasis šios užduoties pabaigos laikas lygus $EF(9) = ES(9) + p_9 = 80$.



4.8 pav. Užduočių vėlyvųjų pradžių (LS) ir pabaigų (LF) nustatymas

Pvz. 5-osios užduoties $LS(5) = ES(7) - p_5 = ES(8) - p_5 = 50 - 35 = 15$, o $LF(5) = LS(5) + p_5 = 50$.



4.9 pav. Projekto kritinio kelio radimas (pažymėtas dvigubomis rodyklėmis). Į grandinę sujungiamos tos užduotys, kurių *ES* = *LS*

4.3.3 Užduočių pirmumo sąrašo sudarymas kritinių kelių metodu

Dažniausiai dėl patogumo projektas būna papildomas dviejomis fiktyviomis užduotimis (PRADŽIA, PABAIGA, arba 0-inė ir (n+1)-oji užduotys, kai turime n užduočių), kurios žymi projekto pradžios ir pabaigos momentus. Jų trukmė laikoma lygi 0 ir šios užduotys neturi jokios įtakos visam projektui. Viršūnės (užduoties) *X kritiniu keliu* vadinamas ilgiausios trukmės kelias (iš visų galimų) iš viršūnės *X* į viršūnę PABAIGA. Ilgiausias kelias iš viršūnės PRADŽIA į viršūnę PABAIGA yra viso projekto kritinis kelias. Kaip jau minėta, kritinis laikas – tai visų užduočių, esančių kritiniame kelyje, trukmių suma.

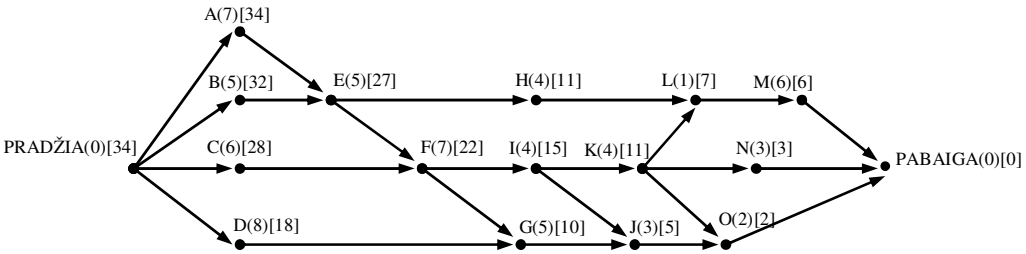
Tam, kad sudaryti užduočių pirmumo sąrašą pagal viršūnių kritinius laikus, reikia kiekvienai viršūnei *X* rasti kritinį kelią ir laiką. Tai galima atlikti *priešsrovio algoritmu* (Tannenbaumas, Arnoldas, 1995) (žiūr. 4.10 pav.):

1. Pradedame nuo viršūnės PABAIGA. Kadangi šios fiktyvios užduoties trukmė yra 0, tai ir kritinis laikas (iki savęs pačios) lygus 0 (KL(PABAIGA) = 0).
2. Bet kurios i-osios užduoties, po kurios seka užduotys j (tame tarpe ir PABAIGA), kritinis laikas $KL(i) = max\{KL(j), (i,j)\in C\} + p_i$.

Remdamiesi kritinio kelio ir kritinio laiko sąvokomis, *sudarome užduočių pirmumo sąrašą*:

1. *Priešsrovio* algoritmu randame kiekvienos i-osios užduoties kritinį laiką $KL(i)$.
2. Visų užduočių sąrašą surikiuojame jų kritinių laikų mažėjimo tvarka. T.y. sudarome sąrašą $b(b_0, b_1, b_2, ..., b_{n+1})$, kur $KL(b_i) \le KL(b_j)$, kai $i < j$.

Taip sudarytas pirmumo sąrašas vadinamas *kritinių kelių sąrašu***.** Tvarkaraščio sudarymo procesas, kuris remiasi šiuo sąrašu, vadinamas *kritinių kelių algoritmu***.**

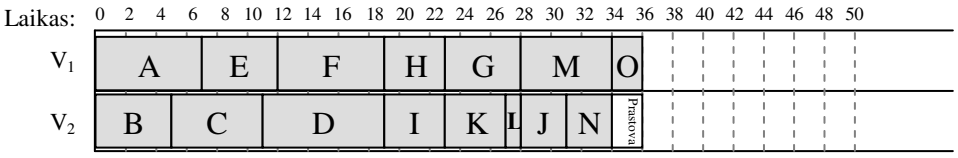


4.10 pav. Priešsrovio algoritmu surasti projekto užduočių kritiniai laikai, kai žinomos užduočių trukmės (viršūnė(trukmė)[kritinis laikas])

Kritinių kelių algoritme pirmumas yra teikiamas užduotims, remiantis visų prieš jas einančių užduočių bendra vykdymo trukme. Jei projektą reikia įgyvendinti per patį mažiausią laiką (t.y. per kritinį laiką), tai būtina visas kritinio kelio užduotis atlikti kiek įmanoma anksčiau. Bet koks uždelsimas pradėti vykdyti bet kurią kritinio kelio užduotį neišvengiamai padidina viso projekto vykdymo trukmę. Kritinis kelias, sudarytas neatsižvelgiant į išteklių ribojimus, nustato apatinę projekto vykdymo ribą, t.y. tokį mažiausią laiką, anksčiau kurio projektas negali būti atliktas.

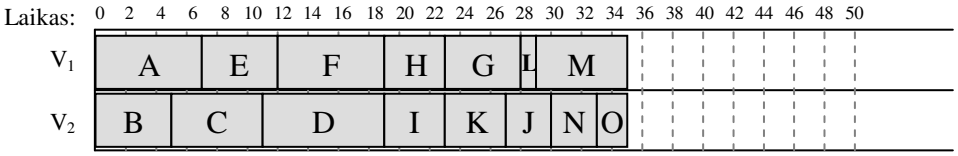
Reikia pastebėti, kad kritinis kelias, rastas atsižvelgiant į išteklių ribojimus, nebūtinai yra optimalus viso projekto vykdymo požiūriu.

Sudarome pirmumo sąrašą pagal kritinius kelius: *A[34], B[32], C[28], E[27], F[22], D[18], I[15], H[11], K[11], G[10], L[7], M[6], J[5], N[3], O[2].*



4.11 pav. Projekto tvarkaraštis, gautas kritinių kelių algoritmu

Naudodamiesi kritinių kelių sąrašu kaip pirmumo sąrašu dviems procesoriams, gausime tvarkaraštį, kuris parodytas 4.11 pav. Tačiau ir šį sprendinį galima pagerinti (žiūr. 4.12 pav.).



4.12 pav. Projekto optimalus tvarkaraštis dviems procesoriams

Kadangi nė vienas procesorius nelieka be darbo per visą vykdymo laiką, tai šis tvarkaraštis yra optimalus.

Kritinių kelių algoritmas yra plačiai naudojamas tvarkaraščių sudarymo metodas, daugeliu atveju pranokstantis mažėjančių trukmių algoritmą, tačiau jis negarantuoja, kad sudarytas tvarkaraštis bus optimalus.

4.4 Išvados

1. RITSU uždavinio optimalus sprendinys bendru atveju gali būti gautas eksponentinio sudėtingumo algoritmais, t.y. pilno perrinkimo, šakų ir rėžių metodais ir pan.
2. Skaičiuojamojo eksperimento metu nustatyta, kad optimalius tvarkaraščius be ribojimų naudojant priimtinus skaičiuojamuosius išteklius galima gauti pilnai perrenkant tik kai $N = 30 \div 40$.
3. Polinominio sudėtingumo algoritmų (kritinių kelių, mažėjančių trukmių) taikymas leidžia gauti sprendinius, kurie naudojami kaip pradiniai artiniai kituose optimizavimo algoritmuose.

5 skyrius. Euristinės paieškos metodų analitinis tyrimas

Daugelį sveikaskaičio arba kombinatorinio optimizavimo uždavinių galima išspręsti tik eksponentinio sudėtingumo metodais. Todėl tokiems uždaviniams spręsti yra taikomos įvairios euristinės paieškos procedūros (Hoos, Stutzle, 2004). Euristinių procedūrų tyrimas bei taikymas tvarkaraščių, ir apskritai, kombinatoriniams ar sveikaskaičiams uždaviniams spręsti priklauso metaeuristinių metodų sričiai. Kiekvieną metaeuristinį metodą sudaro kelių euristinių procedūrų, kurias galima realizuoti įvairiais būdais, seka (Voss, Martello, Osman, Roucairol, 1999, Voss, 2001). Tad kiekvienas metaeuristinis metodas iš tikrųjų yra paieškos paradigma, kurios realizavimas konkrečiu atveju priklauso nuo sprendžiamų uždavinių klasės, skaičiuojamųjų bei žinių išteklių ir pan. Vienos iš žinomiausių tokių metaeuristinės paieškos paradigmų yra modeliuojamojo atkaitinimo (Simulated annealing) algoritmas, paieška su draudimais (Tabu search), išbarstytoji (Scatter search) paieška, genetiniai algoritmai, dalelių spiečių metodai, neuro tinklai ir kt. Metaeuristiniuose algoritmuose paprastai derinamos globaliosios ir lokaliosios paieškos procedūros. Kiekvienoje metaeuristikoje yra įvedamos euristinės procedūros, padedančios „ištrūkti“ iš lokaliųjų optimumų traukos zonų. Aptikus globaliojo ekstremumo traukos zoną, optimalus sprendinys yra tikslinamas lokaliosios paieškos euristinėmis procedūromis. Šiame skyriuje apžvelgsime įvairias metaeuristines paieškos paradigmas, taikomas tvarkaraščiams optimizuoti.

5.1 Godieji algoritmai (greedy algorithms)

Optimizavimo algoritmas vadinamas godžiuoju, jei kiekviename žingsnyje yra parenkamas tik geriausias tame žingsnyje sprendinio variantas, neatsižvelgiant į šio pasirinkimo įtaką tolimesnei sprendimo eigai. Tačiau tokia strategija yra efektyvi tik lokaliosios paieškos požiūriu ir leidžia aptikti lokalųjį ekstremumą, kuris gali skirtis nuo globaliojo ekstremumo. Tolydaus optimizavimo atveju godžių algoritmų pavyzdžiu gali būti gradientinės paieškos algoritmai. Panagrinėsime keletą godžiųjų algoritmų pavyzdžių.

5.1 Pavyzdys. Kuprinės uždavinys (žiūr. 4.1.2 skyrelį). Godusis algoritmas „krauna“ kuprinę pradedant nuo sunkiausiojo nešulio svorių mažėjimo tvarka. Tačiau šiuo algoritmu rastas sprendinys yra nebūtinai optimalus. Tuo galima įsitikinti išnagrinėjus trijų nešulių (15kg, 9kg, 8kg) uždavinį, kai kuprinės talpa yra 20kg.

5.2 Pavyzdys. Artimiausio kaimyno algoritmas. Šis algoritmas, taikomas trumpiausiam ciklui, einančiam per visas grafo viršūnes, rasti, taip pat yra godusis. Kiekviename šio algoritmo žingsnyje į ieškomą kelią yra įtraukiama ta neaplankyta viršūnė, kurios atstumas iki paskutinės aplankytos viršūnės yra trumpiausias, atsižvelgiant į sąlygą, jog nesusidarytų dalinis ciklas. Šiuo algoritmu rastas sprendinys taip pat nebūtinai bus optimalus.

5.3 Pavyzdys. Pigiausios jungties algoritmas. Trumpiausiojo jungiančio medžio paieškos šiuo algoritmu metu pirmiausia į medį yra įtraukiamas lankas su mažiausiu svoriu iš visų grafo lankų. Paskui į medį yra įtraukiami lankai ilgėjimo tvarka, jeigu jie nesukuria ciklų. Šiuo algoritmu yra surandamas trumpiausias jungiantysis medis, tad pigiausios jungties algoritmas yra godaus algoritmo, leidžiančio surasti optimumą, pavyzdys.

Godieji algoritmai dažnai yra taikomi pradiniam sprendiniui rasti metaeuristiniuose algoritmuose.

5.2 Modeliuojamojo atkaitinimo metodas (Simulated annealing)

Šis metodas remiasi analogija su fizikiniu procesu – atkaitinimu (grūdinimu). Atkaitinimo modeliavimo metodas sukurtas remiantis analogijomis iš statistinės mechanikos, modeliuojant procesus sistemose, sudarytose iš didelio skaičiaus mažų diskrečiųjų dalelių. Modeliuojant tokios sistemos „atkaitinimą“, pradžioje sistemai yra suteikiama aukšta temperatūra, kuri palaipsniui mažinama, kol sistema „užsigrūdina“ (pereina į optimalią būseną). Šio modeliavimo pradininkai buvo Metropolis, Rosenbluth ir kiti (Metropolis, Rosenbluth, Rosenbluth, Teller, Teller, 1953). Jie pasinaudojo Bolcmano (eksponentiniu) pasiskirstymo dėsniu apibrėždami tikimybę, kad sistema, įvykus joje tam tikrai perturbacijai, pereis iš vieno energijos lygio (E_1) į kitą (E_2), kai temperatūra lygi t:

$$\mathrm{P} = \begin{cases} 1, \Delta E < 0, \\ \mathrm{e}^{-\Delta E / C_B t}, \Delta E \geq 0, \end{cases} \tag{5.1}$$

čia $\Delta E = E_2 - E_1$, o C_B – Bolcmano konstanta. Kirkpatrick su bendraautoriais (Kirkpatrick, Gelatt, Vecchi, 1983) buvo pirmieji, kurie panaudojo atkaitinimo modeliavimo metodą sprendžiant kombinatorinio optimizavimo uždavinius.

Modeliuojamojo atkaitinimo metodas yra taikomas diskretiems ir tolydiems optimizavimo uždaviniams spręsti (Connolly, 1990, Dorea, 1997, Bouleimen, Lecocq, 1998, Locatelly, 2000a, Locatelly, 2000b, Dzemyda, Senkienė, 1997, Misevičius, 2000, Onbasoglu, Ozdamar, 2001, Bouleimen, Lecocq, 2003, Kim, Moon, 2003). Svarbus šio metodo parametras yra temperatūra, mažėjanti su kiekvienu atkaitinimo proceso imitavimo žingsniu. Šis parametras yra reguliuojamas įvedant specialią temperatūros atnaujinimo (aušinimo) funkciją. Šiame algoritme yra įvedamas specialus nuo temperatūros priklausantis tikimybinis patvirtinimo kriterijus, kuriuo remiantis kiekviename žingsnyje yra priimamas surastas geresnis sprendinys, nei esamas bei su nedidele tikimybe gali būti priimamas blogesnis sprendinys. Ši savybė leidžia „ištrūkti“ iš lokaliųjų optimumo taškų, ieškant globaliojo optimumo (Lundy, Mees, 1986, Gelfand, Mitter, 1993). Šio algoritmo galimybę rasti globalųjį sprendinį galima padidinti, įvedus leistiną minimalų atstumą, kuriuo yra generuojami nauji sprendiniai (Yang, 2000). Atstumo apskaičiavimas priklauso nuo

naudojamos metrikos. Pirmuosiuose algoritmo žingsniuose minimalus atstumas gali būti parenkamas pakankamai didelis, o po to nuosekliai vis mažinamas.

Modeliuojamojo atkaitinimo algoritmui realizuoti yra taikomos įvairios schemos. Šiame darbe nagrinėsime šio algoritmo schemą, pasiūlytą (Yang, 2000)**.**

I. Nustatome (sugeneruojame) pradinį sprendinį *s;* nustatome pradinę temperatūrą *T*, naudojantis atkaitinimo proceso temperatūros atnaujinimo funkcija; nustatome minimalų leistiną atstumą; apskaičiuojama tikslo funkcijos reikšmė $z = f(s)$.

II. Generuojamas kaimyninis sprendinys *s'*. Jei atstumas nuo *s* iki *s'* yra mažesnis, nei duotas minimalus, tai generuojamas naujas kaimyninins sprendinys.

III. Jei naujasis sprendinys yra leistinu atstumu nuo einamojo, tai apskaičiuojama tikslo funkcijos reikšmė $z' = f(s')$.

IV. Taikomas Metropolio patvirtinimo kriterijus:

Jei $\eta < e^{\frac{z-z'}{T}}$, tai priimamas naujasis sprendinys *s'*, priešingu atveju paliekamas einamasis sprendinys *s* (η – atsitiktinis skaičius, tolygiai pasiskirstęs intervale (0,1)).

Šie žingsniai yra kartojami tol, kol netenkinamas koks nors algoritmo stabdymo kriterijus. Tokiu kriterijumi dažniausiai yra maksimalus leistinas žingsnių skaičius.

Skyrelyje 7.2 yra pateikiami rezultatai, taikant šią modeliuojamojo atkaitinimo algoritmo schemą tolydžiajam optimizavimui, o taip pat šios schemos apibendrinimas ir tyrimų rezultatai tvarkaraščių su ribojimais optimizavimo atveju.

Atkaitinimo modeliavimo algoritmo realizacijos gali skirtis leistino atstumo funkcijomis, atkaitinimo („atšaldymo“) schema ir baigimo sąlyga. Atkaitinimo modeliavime temperatūra yra palaipsniui mažinama, pradedant nuo tam tikros pradinės reikšmės t_0. Ši reikšmė neturėtų būti nei per didelė, nei per maža. Iš tikrųjų, jei pradinė temperatūra būtų pernelyg aukšta, tai atkaitinimo procesas užsitęstų labai ilgai, bereikalingai nagrinėjant daug „blogų“ sprendinių. Kita vertus, per daug žema pradinė temperatūra gali lemti patekimą į nebūtinai gero lokaliojo optimumo traukos zoną. Kartais naudojamas pusiausvyros testas temperatūros mažinimo momentui nustatyti. Pusiausvyra apibūdinama stabilios proceso būsenos kriterijumi. Kombinatorinių uždavinių atveju pusiausvyros kriterijumi galėtų būti, pvz., tikslo funkcijos reikšmių svyravimų amplitudė (jeigu ji nedidelė, tai priimama, jog pusiausvyra pasiekta). Atsižvelgiant į pusiausvyros testą, yra skiriami du atkaitinimo tipai: homogeninis ir nehomogeninis. Pirmuoju atveju bandymai atliekami su ta pačia, fiksuota temperatūra, kol pasiekiama pusiausvyra: tada temperatūra sumažinama ir procesas kartojamas vėl. Antruoju atveju temperatūra mažinama po kiekvieno bandymo. Pusiausvyros testas neatliekamas. Kai kuriose atkaitinimo modeliavimo algoritmų versijose temperatūra mažinama ir didinama periodiškai. Tyrimų, atliktų dar 1989 m., rezultatai rodo (Ingber, 1989), kad tikslinga taikyti „atkaitinimų“ ir „kaitinimų“ seką, vadinamą pakartotiniu atkaitinimu (angl. reannealing), nes tai suteikia papildomas, lankstesnes galimybes kuriant atkaitinimo modeliavimo algoritmus.

Modeliuojant atkaitinimą svarbu parinkti tinkamą temperatūros mažinimo formulę. Įvairios temperatūros atnaujinimo funkcijos apžvelgiamos (Kirkpatrick, Gelatt, Vecchi, 1983, Locatelly, 2000b, Yang, 2000).

Atkaitinimo procesas teoriškai turėtų būti tęsiamas tol, kol galutinė temperatūra tampa lygi 0. Tačiau atkaitinimą galima baigti ir anksčiau, kai tampa mažai tikėtina, jog bus pasiektas pagerėjimas. Pvz., kai tikslo funkcijos reikšmė nemažėja pakankamai ilgą laiko tarpą, arba, kai, atlikus pakankamai daug bandymų, priimtų teigiamų sprendimų (t. y., perėjimų iš vieno sprendinio į kitą) skaičius tampa mažesnis už tam tikrą slenkstį. Dažnai kaip baigimo sąlyga tarnauja fiksuotas atliekamų bandymų skaičius, t. y. atkaitinimo schemos „ilgis“.

Panagrinėkime MA metodo taikymą tvarkaraščiams optimizuoti bei galimybę automatiškai optimizuoti parametrų parinkimą. Pradinis tvarkaraštis gerinamas tikrinant įvairius perstatinius. Po kiekvienos iteracijos geriausias surastas tvarkaraštis yra įsimenamas. Perėjimai į blogesnius tvarkaraščius yra atliekami su tam tikromis tikimybėmis. Metodo idėja yra iš einamojo tvarkaraščio i pereiti į gautąjį tvarkaraštį $i+1$ su tikimybe vienetas, jei $h_{i+1} = C_{i+1} - C_i < 0$, kur C_i ir C_{i+1} yra einamojo ir gautojo naujo tvarkaraščio „baudų“ taškai. Jei $h_{i+1} > 0$, tada į tvarkaraštį $i+1$ pereiname su tikimybe r_{i+1},

$$r_{i+1} = \begin{cases} e^{\frac{-h_{i+1}}{x_1 / ln(1+N)}}, \; jei \; h_{i+1} > 0, \\ 1, \; kitu \; atveju. \end{cases} \tag{5.2}$$

Čia N yra iteracijos numeris, o $x_1 > 0$ yra pradinė temperatūra.

Gali kilti klausimas, kokią pradinę temperatūros reikšmę reikėtų parinkti. Tam gali būti naudojamas patobulintas MA. Skirtumas nuo tradicinio MA yra tas, kad tam tikrą fiksuotą iteracijų skaičių $N = K$ yra optimizuojamas parametras x_1. Šiuo atveju temperatūros atnaujinimo (aušinimo) funkcija taip pat turi būti optimizuojama. Įvedamas antrasis parametras $x_2 > 0$. (5.2) formulę pertvarkome į (5.3):

$$r_{i+1} = \begin{cases} e^{\frac{-h_{i+1}}{x_1 / ln(1+x_2 N)}}, \; jei \; h_{i+1} > 0, \\ 1, \; kitu \; atveju. \end{cases} \tag{5.3}$$

Čia $a_k \le x_k \le b_k$, $k = 1, 2$, o apatinės ribos $a_k > 0$ ir viršutinės ribos $b_k > a_k$ yra fiksuotos, atsižvelgiant į preliminarius skaičiavimo rezultatus. Be MA parametrų x_1 ir x_2, pavyzdžiui, užsiėmimų tvarkaraščių uždaviniuose, dar naudojamas perstatinių parametras $0 < x_0 < 1$. Šis parametras nusako tikimybę praleisti tam tikrą asmenį, kaip tai panašiai daroma tradiciniuose mokyklų tvarkaraščių modeliuose. Todėl optimizuojamasis vektorius yra $x = (x_0, x_1, x_2)$ (Mockus, 2000).

5.3 Paieška su draudimais (Tabu search)

Šis metodas remiasi tam tikrų sprendinių uždraudimo idėja, siekiant „ištrūkti“ iš lokaliųjų optimumų traukos zonų. Pagrindinis šio metodo parametras – tabu sąrašas (tabu list), kuris pradžioje būna tuščias, o po to yra papildomas ir modifikuojamas. Į šį sąrašą optimizavimo eigoje yra įtraukiami jau nagrinėti sprendiniai arba jų aplinkos. Tad šiame metode yra draudžiama tam tikrą laiką grįžti prie jau nagrinėtų sprendinių arba jų aplinkų, kol draudimas bus išstumtas iš fiksuoto ilgio ir nuolat kintančio tabu sąrašo. Paieškos su draudimais metodas remiasi išplėsta lokaliąja pieška. Skirtingai, negu lokaliosios paieškos procedūros, kurios apsiriboja lokaliai optimalaus sprendinio (lokaliojo optimumo) suradimu nagrinėjamo sprendinio aplinkoje (kaimyninių sprendinių aibėje), paieška su draudimais pagrįsti algoritmai tęsia paiešką ir tuo atveju, kai surandamas lokalusis optimumas, t. y., kai duoto sprendinio aplinkoje neįmanoma surasti geresnio sprendinio.

Paieška su draudimais pradedama nuo pradinio sprendinio $s^{(0)}$ iš galimų sprendinių aibės S. Pradinis sprendinys gali būti sugeneruotas tiesiog atsitiktiniu būdu. Algoritmo vykdymo metu analizuojama sprendinio $s \in S$ aplinkos (kaimyninių) sprendinių aibė $N(s)$. Baigus analizę, pereinama į tą sprendinį s' iš $N(s)$, kuriam tikslo funkcijos f reikšmė yra mažiausia. Perėjimas atliekamas ir tuo atveju, kai tikslo funkcijos pokytis yra teigiamas (t.y., tikslo funkcijos reikšmė „pablogėja“) – taip galima pereiti nuo vieno lokaliai optimalaus sprendinio prie kito. Grįžimas į anksčiau nagrinėtą sprendinį turi būti uždraudžiamas tam tikram iteracijų skaičiui. Taigi nagrinėtieji sprendiniai yra įtraukiami į tabu sąrašą. Tokiu būdu perėjimas į sprendinį $s' \in N(s)$ yra draudžiamas, jeigu tas sprendinys (ar tam tikras požymis, susijęs su tuo sprendiniu) duotu metu yra tabu sąraše.

Besąlyginis sprendinių draudimas gali apriboti paiešką kai kuriose sprendinių aibės S srityse. Todėl kartu su tabu sąrašu naudojamas aspiracijos kriterijus. Naudojantis šiuo kriterijumi galima anuliuoti „tabu būseną“, esant tam tikroms sąlygoms. Vienas iš standartinių aspiracijos kriterijų yra toks: „perėjimas“ iš sprendinio s į sprendinį s' – nors jis ir yra „tabu“ – leidžiamas, jeigu tenkinama sąlyga $f(s') < f(s^*)$, kur s^* yra geriausias iki šiol paieškos eigoje surastas sprendinys. Paieškos su draudimais algoritmo lankstumui padidinti tabu sąrašas, tiksliau, tabu sąrašo ilgis (dydis) algoritmo vykdymo eigoje gali būti kartas nuo karto atnaujinamas. Paieška tokiu algoritmu užbaigiama, atlikus iš anksto nustatytą paieškos iteracijų (bandymų) skaičių. Tačiau galima įvesti ir kitokias algoritmo stabdymo sąlygas. Paieškos su draudimais algoritmo rezultatas yra geriausiasis surastas sprendinys, kuris nebūtinai sutampa su sprendiniu, gautu paskutinėje paieškos iteracijoje, nes geriausias rastas sprendinys yra įsimenamas tuo momentu, kai jis surandamas.

Paieškos su draudimais algoritmai skiriasi tabu sąrašo tvarkymu, aspiracijos kriterijumi bei kitokiais veiksniais. Pateiksime formalizuotą paieškos su draudimais algoritmo aprašymą.

1. Parenkame pradinį tašką $i_0 \in I$ ir tariame, kad

$f^* := f(i_0)$, $Tabu_L(i_0) := \varnothing$, $k := 0$ (nulinis žingsnis)

2. Kol netenkinamas algoritmo stabdymo kriterijus, vykdyti:

    2.1. Suformuojama aplinka $N_p(i_k)$.

    2.2. Jeigu $N_p(i_k) = \varnothing$, tai $i_{k+1} := i_k$, kitu atveju rasti i_{k+1} tokį, kad

$$f(i_{k+1}) = min \{f(j) \ / j \in N(i_k) \setminus Tabu_L (i_k)\}$$

    2.3. Jeigu $f^* > f(i_{k+1})$, tai $f^* := f(i_{k+1})$.

    2.4. Padidiname k reikšmę vienetu $k := k+1$ ir papildome (atnaujiname) draudimų sąrašą $Tabu_L(i_k)$.

Parametrai p ir L yra duoto algoritmo valdantieji (pagrindiniai) parametrai, kurie atitinkamai reiškia aplinkos didumą ir tabu sąrašo ilgį.

Pažymėsime, jog paieškos su draudimais algoritmą galima įtraukti į kitas metaeuristines procedūras, pavyzdžiui, genetinius algoritmus. Tokiu būdu gautuose hibridiniuose genetiniuose algoritmuose, paieškos su draudimais procedūros pritaikomos generuoti pradinėms aukštos kokybės sprendinių „populiacijoms“ arba geriausiems sprendiniams atrinkti.

Paieškos su draudimais algoritmo pradininkas yra F.Glover, kuris pasiūlė šią lokaliosios paieškos schemą (Glover, 1990, Glover, Laguna, 1997). Šis metodas tapo labai populiarus ir plačiai taikomas įvairiems uždaviniams spręsti (Hanafi, Freville, 1998, Chelouah, Siarry, 2000, Machado, Shiyou, Ho, Peihong, 2001, Alvarez-Valdes, Crespo, Tamarit, 2002, Misevičius, Lenkevičius, 2003). Paieškos su draudimais metodas buvo sėkmingai pritaikytas įvairiems kombinatorinio optimizavimo uždaviniams spręsti – kvadratinio paskirstymo (Skorin-Kapov, 1990), keliaujančio pirklio uždaviniams (Chakrapani, Skorin-Kapov, 1993). Keliaujančio pirklio uždavinyje sprendinį sudarė viršūnių, kurios turi būti apeinamos nuoseklia tvarka, sąrašas. Algoritme panaudotas fiksuoto ilgio (dydžio) tabu sąrašas, kurį sudaro jau perstatytos viršūnės. Algoritmo vykdymo eigoje tas sąrašas tvarkomas pagal taisyklę: „pirmasis įėjo – pirmasis išėjo“ (angl. FIFO – „first input – first output”), t. y., sprendinys (ar to sprendinio požymis), kuris anksčiausiai įtraukiamas į sąrašą, anksčiausiai iš to sąrašo ir pašalinamas. Paieškos procedūra pagrįsta tokia strategija: duotame sprendinyje neleidžiama sukeisti (perstatyti) elementų, priklausančių tabu sąrašui (tai reiškia, jog tie elementai buvo sukeisti ne anksčiau kaip prieš h iteracijų), nebent sukeitus tuos elementus būtų gautas geresnis sprendinys negu geriausias iki tol surastas. Perėjus prie naujo sprendinio (atlikus elementų sukeitimą), sukeistieji elementai įtraukiami į tabu sąrašą, tuo pačiu pašalinant iš to sąrašo anksčiausiai į jį įtrauktą elementų porą – tai porai anuliuojma „tabu būsena“.

Randomizuotosios paieškos su draudimais algoritmą 1991 m. pasiūlė Taillard (Taillard, 1991). Kitaip negu Skorin-Kapov algoritme, Taillard algoritme buvo panaudotas dinamiškai (atsitiktiniu būdu) kintančio ilgio (dydžio) tabu sąrašas. Šiame algoritme yra modifikuota sprendinių elementų įtraukimo į tabu sąrašą ir jų nagrinėjimo tvarka ir yra panaudotas aspiracijos kriterijus: jeigu kuri

nors elementų pora nebuvo sukeista per pakankamai didelį iteracijų skaičių, tai tos poros elementai yra sukeičiami ir pereinama į naują sprendinį, nepriklausomai nuo „tabu būsenos“ ir tikslo funkcijos reikšmės. Svarbi Taillard algoritmo savybė, leidžianti paspartinti paiešką, yra ta, jog einamojo sprendinio (perstatymo) π aplinka $N(\pi)$ išnagrinėjama panaudojant $O(n^2)$ operacijų vietoje $O(n^3)$ operacijų Skorin-Kapov algoritme. Sukurta ir kitų paieška su draudimais besiremiančių algoritmų.

5.4 Genetiniai algoritmai (Genetic algorithms)

Genetinius algoritmus (GA) ir jų sudarymo principus XX-ojo amžiaus 70-aisiais metais pasiūlė Holland (Holland, 1975). Pradedant Holland pirmuoju darbu, genetiniai algoritmai buvo sėkmingai išbandyti, sprendžiant įvairius optimizavimo uždavinius, pavyzdžiui, tokius, kaip elektroninių komponentų išdėstymas, operacijų planavimas ir daugelį kitų (Minga, 1986, Goldberg, 1989, Hartmann, 1998, Khouja, Michalewicz, Wilmot, 1998, Hartmann, 2002, Chelouah, Siarry, 2003, Wang, 2003, Iyer, Saxena, 2004). Genetinių algoritmų metodai pasižymi paprastumu bei universalumu. Genetinių algoritmų veikimas yra pagrįstas evoliucijos, vykstančios gyvojoje gamtoje, t.y., natūraliosios atrankos proceso imitavimu. Pagrindinės sąvokos, kurios naudojamos modeliuojant biologinės evoliucijos procesus, yra „individas“ ir „populiacija“. „Individas“ yra tam tikras elementarus, daugiau neskaidomas vienetas, objektas. Didesnė ar mažesnė „individų“ grupė sudaro „populiaciją“. Dar vienas svarbus dalykas yra vadinamasis „individo tinkamumas“ – savotiška „individo vertė“. „Individo vertę“ galima traktuoti kaip „individo“ sugebėjimo sėkmingai prisitaikyti (prie aplinkos), išlikti ir reprodukuotis laipsnį. Šia prasme „vertingesnis“ (grynai biologiškai) yra tas „individas“, kuris sugeba geriausiai prisitaikyti (būti „stipresnis“ už kitus) ir, gal būt, palikti didesnį palikuonių („vaikų“) skaičių.

Optimizavime vietoje sąvokų „individas“, „populiacija“, „individo vertė“ naudojamos tradicinės, įprastos sąvokos: „individui“ atitinka atskiras sprendinys, „populiacijai“ – sprendinių aibė (grupė, rinkinys), pagaliau, „individo vertė“ yra asocijuojama su tikslo funkcijos reikšme duotajam sprendiniui. Taip, kaip gyvosios gamtos evoliucijoje išlieka tik „vertingiausi“ („stipriausi“) „individai“, taip ir optimizuojant siekiama gauti kuo „geresnį“ sprendinį, t.y., sprendinį su kuo mažesne (ar didesne) tikslo funkcijos reikšme (žiūrint, koks optimizavimo uždavinys – minimizavimo ar maksimizavimo – yra sprendžiamas).

Genetiniuose algoritmuose individai yra vaizduojami sprendiniais, kurie yra koduojami simbolių eilutėmis – chromosomomis. Kiekvienas optimizuojamos funkcijos nežinomas kintamasis yra užkoduojamas simbolių, vaizduojančių genus, rinkiniu, kurie yra apjungiami į chromosomas. Optimizavimo metu yra ieškoma ne vieno galimo sprendinio, o jų aibės, t.y. sprendžiant uždavinį yra operuojama ne su viena chromosoma, o su jų populiacija.

Genetiniai algoritmai apima metaeuristinių metodų šeimą, kuriai būdingos tokios savybės:

1. Operuojama su viena ar keletu leistinų sprendinių „populiacijų“.
2. Atskiri sprendiniai iš vienos ar kelių „populiacijų“ parenkami tolimesniam nagrinėjimui, apdorojimui, teikiant pirmenybę tiems sprendiniams, kurie turi „geresnes“ tikslo funkcijos reikšmes.
3. Naudojamos euristinės procedūros, skirtos naujiems leistiniems sprendiniams formuoti, dalinant gautą populiaciją į sprendinių poras ir operuojant su šiomis poromis.
4. Naudojamos taisyklės naujiems sprendiniams generuoti iš atskirų anksčiau gautų sprendinių.

Savybių (2)–(4) realizavimas leidžia populiaciją atnaujinti iteraciniu būdu, paliekant joje „geresniuosius“ sprendinius.

Formalizuojant genetinių algoritmų aprašymą, aukščiau minėtos bendrosios savybės susiejamos su atitinkamomis euristinėmis procedūromis. Taip (2) savybė susiejama su atrankos procedūra, (3) savybė vadinama kryžminimu, (4) savybė yra vadinama mutavimu.

Yra žinoma daug įvairių būdų, kaip atlikti „sprendinių-tėvų“ parinkimo, kryžminimo, mutavimo bei populiacijos sprendinių atnaujinimo procedūras (Glibovec, Medvidj, 2003). Pavyzdžiui, galima operuoti su viena didele sprendinių populiacija arba su keliomis mažesnėmis („lygiagrečiomis“, „sub-populiacijomis“ ir pan.). „Sprendinių-tėvų“ poros gali būti parenkamos, atsižvelgiant į jų tinkamumą, t. y., tikslo funkcijos reikšmę, ir į galimybes gauti „vertingus“ „sprendinius-palikuonis“. Kryžminimo bei mutavimo procedūros gali būti atliktos įvairiais būdais, beje, jos gali būti apjungtos į vieną procedūrą. Atrankos metu galima pakeisti visus einamosios populiacijos narius (sprendinius) naujai gautais „palikuonimis“ arba taikyti įvairius atrankos metodus, pavyzdžiui, turnyrinę, ranginę, proporcingumo atranką ir pan. Svarbu pažymėti, kad genetinio algoritmo realizacija labai priklauso nuo sprendinių kodavimo būdo.

Detaliau aptarsime genetinio algoritmo paradigmos realizavimo etapus.

Genetinė paieška pradedama sudarant pradinę populiaciją. Dažniausiai pradinė populiacija parenkama atsitiktinai, o po to ji yra optimizuojama kituose algoritmo žingsniuose, naudojant įvairias euristines procedūras. Klasikinėse genetinio algoritmo versijose sprendiniai-chromosomos yra koduojamos binariniu kodu. Genu yra laikomas vienas dvejetainis simbolis arba jų grupė, koduojantys vieną kintamąjį arba požymį.

Sudarius pradinę populiaciją, su ja atliekamos operacijos, modeliuojant evoliucinį procesą, kurio svarbiausi principai yra paveldimumas, mutacija ir atranka. Populiacijos evoliucionavimas yra modeliuojamas remiantis tokiomis taisyklėmis:

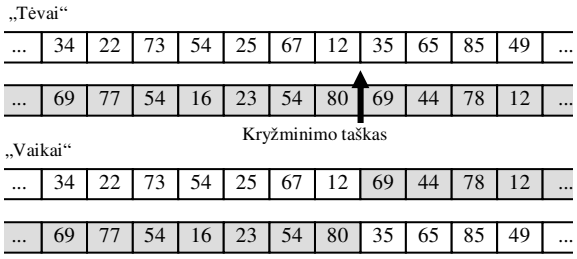
1. Kiekvienam populiacijos chromosomų genui taikomas mutacijos operatorius, kuris su tam tikra tikimybe gali pakeisti geną.
2. Chromosomos suskirstomos poromis, kurios apsikeičia savo genų rinkinių dalimis, t.y. jos yra kryžminamos.

3. Mutavimo ir kryžminimo būdu gautos chromosomos sudaro tarpinę populiaciją. Tikimybė chromosomai dalyvauti formuojant tarpinę populiaciją yra tuo didesnė, kuo aukštesnė ją atitinkanti tikslo funkcijos reikšmė.
4. Iš tarpinės populiacijos sudaroma nauja populiacija, atrenkant „geresnius“ individus, kartojant su ja aukščiau aprašytas operacijas.

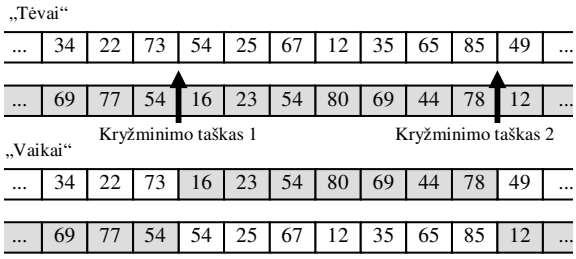
Pagrindinės mutacijos ir kryžminimo funkcijos yra vertingos informacijos, esančios sprendinyje, perėmimas. Konkreti algoritmo realizacija turi užtikrinti potencialią galimybę gauti bet kokį sprendinį iš visų galimų sprendinių aibės, pasinaudojant tik mutacija ir kryžminimu.

Kryžminimas leidžia efektyviai perrinkti didelį kiekį variantų (kadangi pradiniai sprendiniai buvo suformuoti atsitiktiniu būdu). Dažniausiai genetiniuose algoritmuose naudojama „dviejų tėvų“ schema, tačiau „palikuonių“ chromosomos gali būti generuojamos iš daugiau negu dviejų chromosomų. Yra naudojami įvairūs kryžminimo tipai: vieno taško, dviejų taškų, *m*-taškis ir vienalytis (homogeninis). Vieno taško kryžminime išrenkamos dvi chromosomos ir sugeneruojamos atsitiktinis kryžminimo taškas. Šis procesas pavaizduotas (5.1 pav). Dviejų taškų kryžminimas (5.2 pav.) skiriasi nuo vienataškio tik tuo, kad atsitiktinai parenkami du kryžminimo taškai ir chromosomos apsikeičia dalimis tarp tų taškų. *m*-taškio kryžminimo atveju yra parenkama *m* kryžminimo taškų. Šiais taškais chromosomos suskaldomos į *m*+1 dalių ir apsikeičia tiktai lyginėmis arba nelyginėmis jų dalimis. Vienalyčio kryžminimo (5.3 pav.) metu kiekvienas pirmojo tėvo genas turi 50% šansą apsikeisti vietomis su atitinkamu antrojo tėvo genu.

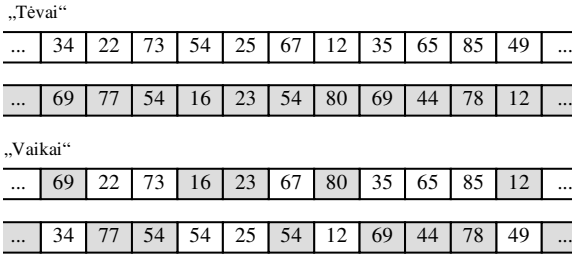
5.1–5.3 paveikslėliuose paryškinami chromosomų genai, parinkti kryžminimui. „Palikuonių“ chromosomos suformuojamo iš „tėvų“ chromosomų genų mainų būdu.



5.1 pav. Vieno taško kryžminimas



5.2 pav. Dviejų taškų kryžminimas



5.3 pav. Vienalytis kryžminimas

Mutacija yra genetinis operatorius, operuojantis tik su viena chromosoma. Mutacijos operatorius pakeičia vieną chromosomos geną atsitiktiniu būdu. Jis yra pritaikomas visiems chromosomos genams. Mutacijos rezultato efektyvumą apsprendžia gautą sprendinį atitinkanti tikslo funkcijos

reikšmė. Kitas genetinis operatorius, operuojantis tik su viena binariniu kodu koduota chromosoma, neturintis analogo gamtoje, yra inversija. Inversijos metu yra išrenkamos dvi atsitiktinės pozicijos chromosomoje ir genai, esantys tarp jų, yra invertuojami. Pagrindinė mutacijos ir inversijos užduotis yra suteikti galimybę perrinkti įvairius sprendinius ir/arba pereiti prie tų sprendinių, kurių neįmanoma gauti kryžminimo būdu. Kita vertus, mutacijos neturi sukelti chaoso ir išsigimimo populiacijoje, kurie gali įvykti, jei mutacijos tikimybė yra didelė. Sprendiniai, atsiradę nesėkmingų mutacijų metu, yra eliminuojami atrankos metu, o sėkmingi sprendiniai tur daugiau šansų išlikti taip pat dėka atrankos.

Atranka, kuri yra svarbus evoliucinio proceso etapas, užtikrina chromosomų dalyvavimą formuojant būsimas kartas. Dalyvavimo tikimybės yra nevienodos ir priklauso nuo chromosomos vertės, kurią labiausiais apsprendžia chromosomą atitinkanti tikslo funkcijos reikšmė. Todėl sprendiniai (chromosomos), kuriuos atitinka „geresnė“ tikslo funkcijos reikšmė, turi daugiau šansų dalyvauti formuojant tarpinę populiaciją. Priklausomai nuo atrankos operatorių, tarpinės populiacijos formavime gali dalyvauti ir sprendiniai, kuriuos atitinka „blogesnės“ tikslo funkcijos reikšmės, siekiant, kad šių sprendinių atranka gali lemti labai „gerų“ sprendinių atsiradimą tolesnės evoliucijos metu.

Išskiriami keli atrankos būdai: turnyrinis, atmetimo, lyginamasis, ranginis, proporcingasis ir kiti. Turnyrinėje atrankoje atsitiktinai atrenkama t chromosomų tarpinėje populiacijoje, iš kurių išrenkamos geriausios. Dažniausiai turnyrai vyksta tarp dviejų individų (binariniai turnyrai, $t = 2$). Tokios atrankos metu populiacijos nereikia rūšiuoti, todėl šis algoritmas yra $O(n)$ sudėtingumo ir yra lengvai realizuojamas (n – individų skaičius populiacijoje). Atmetimo atrankos metu yra išrenkami ir paliekami populiacijoje t geriausių individų. Kadangi tokiam atrankos algoritmui būtina surūšiuoti visą populiaciją, jos sudėtingumas yra $O(n\ ln(n))$. Proporcingoji atranka genetiniuose algoritmuose buvo panaudota pirmiausia (Holland, (1975)). Galimybė būti išrinktam kiekvienam individui yra tiesiogiai proporcinga jį atitinkančios tikslo funkcijos reikšmei. Šio algoritmo sudėtingumas yra $O(n)$. Proporcingoji atranka turi godaus algoritmo savybių, todėl jos metu gali būti praleisti sprendiniai, esantys globaliojo optimumo traukos zonoje. Ranginę atranką pasiūlė Baker (Baker, 1985), siekiant išvengti proporcingumo algoritmo trūkumų. Tokios atrankos metu chromosomos rūšiuojamos pagal vertę, priskiriant „geriausioms“ iš jų rangą N, o „blogiausioms“ – rangą 1. Kitoms yra priskiriami tarpiniai rangai. Tikimybė būti išrinktam ir likti tolesnėje evoliucijoje yra tiesiogiai proporcinga chromosomos rangui. Kituose algoritmo variantuose išlikimo tikimybė gali priklausyti nuo rango eksponentiškai. Ranginiai atrankai taip pat reikia rūšiuoti populiaciją, todėl jos sudėtingumas taip pat yra $O(N\ ln\ N)$.

Genetiniuose algoritmuose susiduriama su „klonavimo“ problema, kai populiacijoje atsiranda daug vienodų „individų“. Tinkamai parinkus metodo parametrus, genetinis algoritmas turi užtikrinti galimybę perrinkti visus įmanomus sprendinius, išvengiant didelio skaičiaus „klonų“. Tačiau

vykdymo metu siekiama kuo greičiau aptikti optimalaus sprendinio traukos zoną ir patį sprendinį. Todėl genetiniuose algoritmuose yra svarbi algoritmo stabdymo ir galutinio sprendinio nustatymo problema. Dažniausiai genetiniuose algoritmuose sustojama peržiūrėjus tam tikrą sprendinių (chromosomų) skaičių. Taip pat paiešką galima stabdyti, jei geriausia tikslo funkcijos reikšmė populiacijoje nesikeičia per nustatytą žingsnių skaičių arba jos reikšmė pasiekia tam tikrą, iš anksto nustatytą, reikšmę.

5.5 Išbarstytoji paieška (Scatter search)

Tai evoliucinis metodas, naudojamas kombinatorinio ir netiesinio optimizavimo uždaviniams spręsti (Glover, 1998, Glover, Laguna, Martí, 2003). Išbarstytoji paieška operuoja sprendinių rinkiniu, vadinamu atraminiu. Kombinuojant to rinkinio sprendinius, yra gaunami nauji sprendiniai. Sprendiniai į atraminį rinkinį atrenkami, naudojant procedūras, panašias į taikomas genetiniuose algoritmuose. Skirtingai nuo genetinių algoritmų, išbarstytoji paieška, operuoja su žymiai mažesniu sprendinių rinkiniu. Mat genetiniuose algoritmuose nauji sprendiniai yra gaunami kombinuojant nagrinėjamos populiacijos elementų poras. O kombinuojant sprendinius išbarstytoje paieškoje, kai jie imami iš atraminių sprendinių rinkinio įvairiais poaibiais, galima gauti pakankamai daug įvairių kombinacijų. Todėl pakanka operuoti su nedideliu sprendinių rinkiniu. Reikia pažymėti, kad išbarstytosios paieškos metu gali būti nagrinėjami neleistini arba nepasiekiami sprendiniai. Nepasiekiamais sprendiniais yra vadinami sugeneruoti sprendiniai, kurie nepriklauso uždavinio galimų sprendinių aibei. Pavyzdžiui, sveikaskaičio optimizavimo uždavinio sprendinio išbarstytosios paieškos metu gali būti sugeneruotas sprendinys su neigiamomis ar trupmeninėmis reikšmėmis. Toks sprendinys yra vadinamas nepasiekiamu. Toks sprendinys gali būti paverčiamas galimu sprendiniu, pvz. jį suapvalinus.

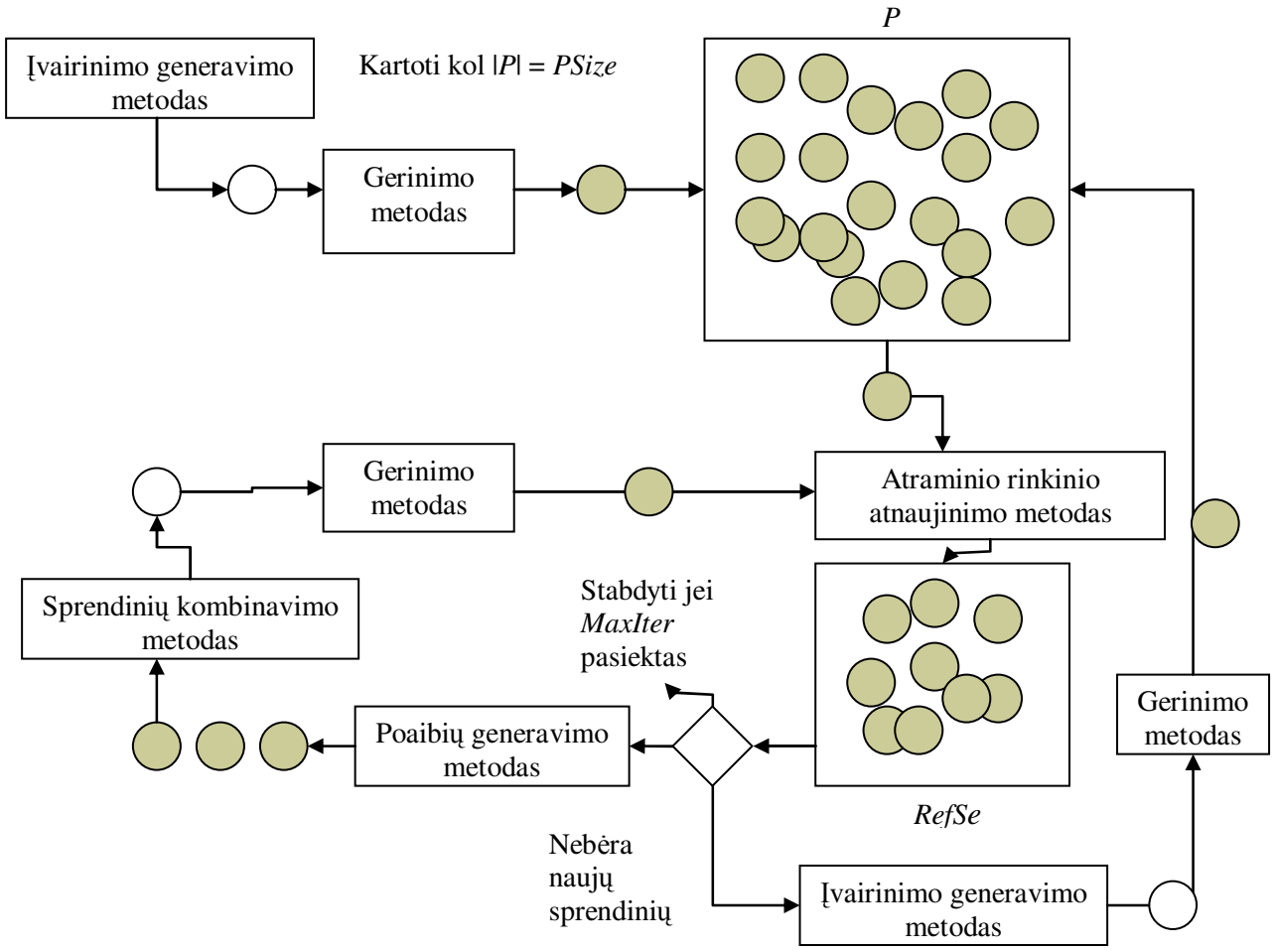
Išbarstytosios paieškos yra realizuojami keliais etapais, vadovaujantis tokiomis taisyklėmis.

1. Paieška pradedama pradinio sprendinių rinkinio generavimu, užtikrinant jo įvairovės lygį ir gerinimo galimybes.
2. Nauji sprendiniai yra generuojami, naudojant esamo atraminio sprendinių rinkinio poaibių iškilą apvalkalą. Reikia pastebėti, kad nauji sprendiniai gali būti nebūtinai leistini ir pasiekiami.
3. Sprendiniai, gauti antrame žingsnyje, yra modifikuojami, dalį jų paverčiant leistinais ir pasiekiamais.
4. Iš generuotų ir po to modifikuotų sprendinių, yra atrenkami „geriausi“, kurie yra priskiriami atraminiam rinkiniui. Sprendinio vertė priklauso nuo jį atitinkančios funkcijos reikšmės bei nuo galimybės gauti efektyvius sprendinius tolesnės evoliucijos metu.

Išbarstytoji paieška yra stabdoma atlikus nustatytą iteracijų skaičių.

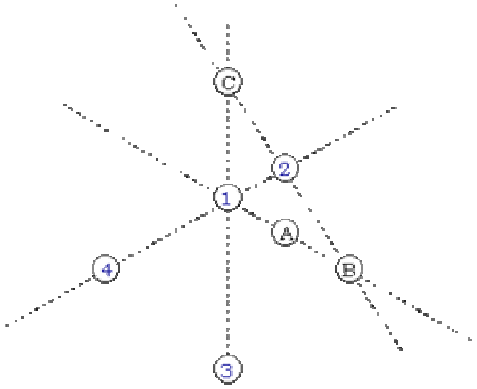
Panagrinėsime atskirus išbarstytosios paieškos etapus (žiūr. 5.5 pav.).

1. **Įvairovės didinimo procedūra**. Tokios procedūros naudojamos gauti pradinį įvairių (skirtingų) sprendinių rinkinį.
2. **Gerinimo (patobulinimo) procedūra.** Ji naudojama sprendiniams transformuoti į vieną ar kelis geresnius („sustiprintus“) sprendinius.
3. **Atraminio sprendinių rinkinio atnaujinimo (papildymo) procedūra**. Ji naudojama sudaryti atraminį sprendinių rinkinį iš b „geriausių“ sprendinių. Atraminių sprendinių skaičius b yra nedidelis, pvz. ne daugiau 20. Tokio kiekio sprendinių dažniausiai pakanka efektyviai realizuoti kitus išbarstytosios paieškos metodo etapus. Sprendiniai įtraukiami į tą atraminį sprendinių rinkinį priklausomai nuo juos atitinkančios tikslo funkcijos reikšmės ir jų skirtingumo.
4. **Poaibių generavimo procedūra.** Šios procedūros metu yra generuojami atraminio sprendinio iškilo apvalkalo sprendinių poaibiai.
5. **Sprendinių generavimo procedūra.** Šios procedūros metu yra formuojami nauji sprendiniai, pasinaudojant sugeneruotais jų poaibiais.



5.5 pav. Išbarstytosios paieškos schema (Laguna, 2006)

Bendru atveju, poaibiai, naudojami naujiems sprendiniams generuoti, gali būti sudaryti iš įvairaus skaičiaus sprendinių. Dažniausiai naujiems sprendiniams generuoti yra naudojamos visos atraminio rinkinio sprendinių poros. Tad skirtingai nuo genetinių algoritmų, kuriuose populiacija padalinama į $n/2$ porų, išbarstytosios paieškos metu yra išnagrinėjamos visos $n\,(n\text{-}1)/2$ atraminio rinkinio poros. Nauji sprendiniai gali būti gaunami kitų dviejų sprendinių tiesine kombinacija. Jeigu uždavinys yra sveikaskaitis ir tokiu būdu gautas sprendinys yra nepasiekiamas, jis yra paverčiamas pasiekiamu, pvz. apvalinant. 5.6 pav. yra parodytas atraminio rinkinio, kurį sudaro sprendiniai, pažymėti raidėmis A, B ir C, plėtojimasis. Matome, kad su sprendinių A ir B tiesine kombinacija yra gaunamas sprendinys, pažymėtas 1. Šis sprendinys yra parenkamas pagal tam tikrą kriterijų sprendinių aibėje, esančioje tiesėje, kuri eina per taškus A ir B. Panašiai iš B ir C yra gaunamas sprendinys 2, o iš sprendinių C ir 1 yra gaunamas 3 ir t.t. Tokiu būdu gautame tarpiniame rinkinyje yra 7 sprendiniai, iš kurių yra sudaromas naujas atraminis sprendinys, remiantis tam tikrais kriterijais.



5.6 pav. Taškų rinkinys dvimačiu atveju

5.1 Lentelė. Genetinio algoritmo (GA) ir išbarstytosios paieškos (scatter search, SS) palyginimas

	GA	**SS**
Populiacija	Didelė (~ 100)	Maža (~10)
Reprodukcija	Tikimybinis „tėvų“ parinkimas	Deterministinis atraminių sprendinių parinkimas
Sprendinių kombinavimas	Kryžminimas ir mutacija	Struktūrinės kombinacijos
Evoliucija	Išlikimas po tinkamumo testo	Strateginis atnaujinimas kokybei ir įvairovei palaikyti
Lokalioji paieška	Pridėta kaip mutacijos mechanizmas	Neatskiriama procedūros dalis

5.1 lentelėje yra lyginamos genetinių algoritmų ir išbarstytosios paieškos savybės.

5.6 Kintamos aplinkos paieškos metodas (Variable neigborhood search)

Kintamos aplinkos paieškos (KAP) metodas yra pakankamai naujas metaeuristinis metodas (Hansen, Mladenovic, 1999). Šis metodas grindžiamas tokiais pastebėjimais:

1. Lokalusis minimumas kurioje nors aplinkoje vienos topologinės erdvės požiūriu nebūtinai bus lokalusis minimumas kitoje topologinėje erdvėje.
2. Globalusis minimumas sutampa su sprendiniu, kurį atitinka lokalusis minimumas visų galimų aplinkų struktūrų požiūriu.
3. Daugelyje uždavinių lokalieji minimumai vienos ar kelių aplinkų struktūrų požiūriu yra palyginus arti vienas kito.

Pastarasis pastebėjimas yra empirinis ir pastebėtas sprendžiant daugelį uždavinių. Iš jo išplaukia, jog lokalusis optimumas dažnai suteikia informaciją apie globalųjį optimumą. Gauto lokaliojo optimumo aplinka yra tiriama, siekiant „ištrūkti“ iš lokaliojo optimumo traukos zonos ir surasti kitą lokalųjį optimumą su geresne tikslo funkcijos reikšme.

Skirtingai nuo kitų metaeuristikų, pagrindiniai KAP etapai yra pakankamai paprasti. KAP yra susijusi su nesudėtingų operacijų sekomis, kurios gali padėti rasti efektyvesnius sprendinius.

5.7 Metaeuristikų klasifikacija

M.Laguna (Laguna, 2006) pasiūlė metaeuristikų *x/y/z* klasifikacijos schemą. Ši klasifikacija remiasi trimis metodo parametrais:

- $x = A$ (yra adaptyvi atmintis) arba B (Be atminties),
- $y = N$ (metodiška paieška aplinkoje) arba S (atsitiktinė atranka),
- $Z = 1$ (vienas einamasis sprendinys) arba P (sprendinių populiacija).

Pavyzdžiui, keletą metaeuristikų galima priskirti tokioms klasėms:

- Paieška su draudimais – *A/N/1*,
- Modeliuojamojo atkaitinimo metodas – *B/S/1*,
- Genetiniai Algoritmai – *B/S/P*,
- Išbarstytoji paieška – *B/N/P*.

Išnagrinėjus kelių metaeuristikų taikymo aspektus tam tikros klasės uždaviniams spręsti, galima konstruoti kitų metaeuristikų realizavimo schemas tos klasės uždaviniams spręsti. Pavyzdžiui, modeliuojamojo atkaitinimo ir genetinių algoritmų pritaikymas tvarkaraščių uždaviniams su ribojimais spręsti, leidžia sukurti euristines metodiškos arba atsitiktinės paieškos procedūras su vienu sprendiniu arba su sprendinių populiacija, kurios gali būti pritaikytos realizuojant kitą metaeuristiką, pvz. išbarstytąją paiešką.

5.8 Išvados

Euristinės paieškos metodų analitinis tyrimas leidžia išskirti tokias pagrindines šių metodų savybes.

1. Metaeuristiniuose algoritmuose yra derinamos globaliosios ir lokaliosios paieškos procedūros, siekiant „ištrūkti“ iš lokaliųjų optimumų traukos zonų, ir tikslinant sprendinį lokaliosios paieškos euristinėmis procedūromis aptikus globaliojo ekstremumo traukos zoną.
2. Metaeuristinis algoritmas turi garantuoti principinę galimybę perrinkti visus galimus sprendinius, išvengiant ženklaus tų pačių sprendinių kartotinio nagrinėjimo.
3. Metaeuristiniai algoritmai gali būti klasifikuojami priklausomai nuo sprendinių populiacijos, nagrinėjamos vienoje iteracijoje, dydžio, sprendinio aplinkos tyrimo intensyvumo bei galimybės išsaugoti ir pritaikyti informaciją, gautą optimizavimo proceso metu.

6 skyrius. Stochastiniai ir metaeuristiniai tvarkaraščių optimizavimo metodai

Šiame skyriuje panagrinėsime tvarkaraščių sudarymo atsitiktinės arba euristinės paieškos būdu metodus. Remiantis metaeuristikų klasifikacija, pateikta 5.7 skyrelyje, metaeuristinius algoritmus galima skirstyti į du tipus: nuoseklios paieškos, kai kiekvienoje iteracijoje apskaičiuojama ir realizuojama tik viena tikslo funkcijos reikšmė, ir populiacinius paieškos algoritmus, kai kiekvienoje iteracijoje yra nagrinėjama sprendinių aibė (populiacija). Išnagrinėsime du būdingus šiems tipams algoritmus – modeliuojamojo atkaitinimo (nuosekli paieška) bei genetinį (evoliucinė paieška) algoritmus. Nagrinėsime šių algoritmų realizavimo ypatybes, pasinaudodami tvarkaraščių kodavimu užduočių pirmumo sąrašu, įvestu 3.9 skyrelyje.

6.1 Tvarkaraščių kodavimas ir dekodavimas užduočių pirmumo sąrašu

Nuo sprendinio kodavimo priklauso optimizavimo algoritmo efektyvumas. Koduojant tvarkaraštį darbų pradžių ir pabaigų vektoriais, optimalaus tvarkaraščio radimas suvedamas į daugelio ekstremumų tolydaus optimizavimo su ribojimais uždavinį. Tačiau šio uždavinio sprendimas tolydaus optimizavimo metodais yra labai sudėtinga problema, susijusi su daugelio lokaliųjų ekstremumų perrinkimu. Siekdami palengvinti visų sprendinių bei lokaliųjų ekstremumų perrinkimą, nagrinėsime tvarkaraščio optimizavimo metodus, kurie remiasi tvarkaraščio kodavimu užduočių pirmumo sąrašu. Šio sąrašo panaudojimas leidžia sukurti metodus sprendinių aplinkoms konstruoti bei pereiti nuo vieno sprendinio prie kito.

Aktyviuoju yra vadinamas tvarkaraštis, kuriame užduotis pradedama vykdyti iš kart, jei tai leidžia pirmumo sąrašas ir tam neprieštarauja nuoseklumo sąrašas. Leistinu pirmumo sąrašu vadinsime tokį, kuriam galima rasti užduočių pradžių laikų vektorių, suderintą su nuoseklumo sąryšiais. Remiantis nuoseklumo sąryšių aibe, galima nustatyti sprendinio leistinumą. Pirmumo sąrašo leistinumo patikrinimo algoritmas yra pateikiamas sekančiame skyrelyje.

Žinodami užduočių pradžių laikų vektorių *s*, galime nustatyti užduočių pirmumo sąrašą $b=(0,b_1,b_2,...,b_n,b_{n+1})$. Šiame sąraše užduotys yra išdėstytos pagal atlikimo pirmumą, t.y. $s_{b_i} \le s_{b_j}$, jei $i < j$. Tačiau taip pat galima nagrinėti atvirkščią procedūrą: bet kuriam leistinam užduočių pirmumo sąrašui, pritaikius dekodavimo procedūrą, galima nustatyti aktyvųjį užduočių pradžių vektorių, kuriame darbai prasideda iš eilės, nustatomos pirmumo sąrašu. Nuoseklaus dekodavimo procedūra yra aptariama 6.3 skyrelyje.

Tvarkaraščio optimizavimo algoritmas turi būti konstruojamas taip, kad rasti užduočių pirmumo sąrašą, kurį atitinka optimalus uždavinio (3.1, 3.2) sprendinys.

6.2 Užduočių sąrašo leistinumo nustatymas

Panagrinėsime užduočių sąrašo leistinumo nustatymą. Tegul nuoseklumo sąryšiai užduočių aibėje J nusakomi porų aibe $C = \{(i, j) \mid i \text{ vykdomas prieš } j\}$, $i, j \in J$. Užduočių nuoseklumo sąryšius galime nusakyti binarine matrica $D = \{d_{ij}, 1 \le i, j \le n\}$, čia $d_{ij} = 1, jei (i, j) \in C, d_{ij} = 0, jei (i, j) \notin C$. Galime įvesti pilnąją sąryšių matricą $G = \{g_{ij}, 1 \le i, j \le n\}$, nusakančią visas sąryšių grandines, t.y. $g_{kj} = 1$, jei galima rasti tokią indeksų porų seką $(k, k_1) \in C$, $(k_1, k_2) \in C$, ..., $(k_l, j) \in C$. Matrica D pasižymi tokia savybe, jog $d_{ij} = 1 \Rightarrow d_{ji} = 0$. Matrica G taip pat pasižymi šia savybe. Tad pirmumo sąrašas yra leistinas, jeigu bet kuriai porai $(b_i, b_j) \in b, i < j \Rightarrow g_{b_j, b_i} = 0$. Pilnajai sąryšių matricai nustatyti yra siūloma naudoti tokį formalizuotą algoritmą:

```
FOR i=1 TO n DO
   FOR j=1 TO n DO g(i,j)=d(i,j);
FOR i=1 TO n DO
   FOR j=1 TO n DO
      IF g(i,j)=1 THEN
                  FOR k=1 TO n DO
                    IF g(j,k)=1 THEN g(i,k)=1;
```

Šio algoritmo sudėtingumas yra $O(n^3)$.

6.3 Nuoseklus sąrašo dekoderis

Panagrinėsime nuoseklaus sąrašo dekodavimo procedūrą. Ši procedūra, remdamasi užduočių pirmumo sąrašu, apskaičiuoja ankstyvuosius užduočių atlikimo pradžios momentus, atsižvelgiant į nuoseklumo sąryšius ir išteklių ribojimus. Tokiu būdu gautame tvarkaraštyje nei viena užduotis negali būti pradėta anksčiau nustatyto laiko nepažeidus nuoseklumo sąryšių arba išteklių ribojimų. Kaip jau buvo minėta, tokie tvarkaraščiai vadinami aktyviaisiais. Yra žinoma, kad optimalusis tvarkaraštis taip pat yra aktyvusis. Aktyviajam tvarkaraščiui nustatyti sudarysime tokį algoritmą.

1) Nustatome $s_0 = c_0 = 0$. Tegul i=1.

2) Sakykime, yra nustatyti pirmųjų (i-1) pirmumo sąrašo užduočių pradžios laikai. Tad yra žinomi pradžių ir pabaigų laikai s_{b_j}, c_{b_j}, $j \le i-1$.

Pažymėkime $T_i = \max(\max\limits_{\substack{d_{b_l, b_i} = 1 \\ 0 \le l \le i-1}} c_{b_l}, \max\limits_{0 \le l \le i-1} s_{b_l})$

Sekančios užduoties pradžios laikas yra lygus šiam momentui, jei tenkinami ribojimai ištekliams:

$$s_{b_i} = T_i, \text{ jei } \sum_{\substack{1 \le j \le i-1 \\ s_j \le c_{b_j} \le c_j}} r_{k_j} \le R_k, 1 \le k \le K .$$

Jei ribojimai netenkinami, tai šios užduoties pradžios laikas yra lygus pirmos pasibaigusios užduoties laikui, kai ribojimai jau pradedami tenkinti:

$$s_{b_i} = \min_{\substack{\sum_{\substack{1 \le l \le i-1 \\ s_l \le c_{b_j} \le c_l}} r_{kl} \le R_k, 1 \le k \le K, \; c_{b_j} > T_i, 1 \le j \le i-1}} c_{b_j}$$

Šios užduoties pabaigos laikas nustatomas paprastai:

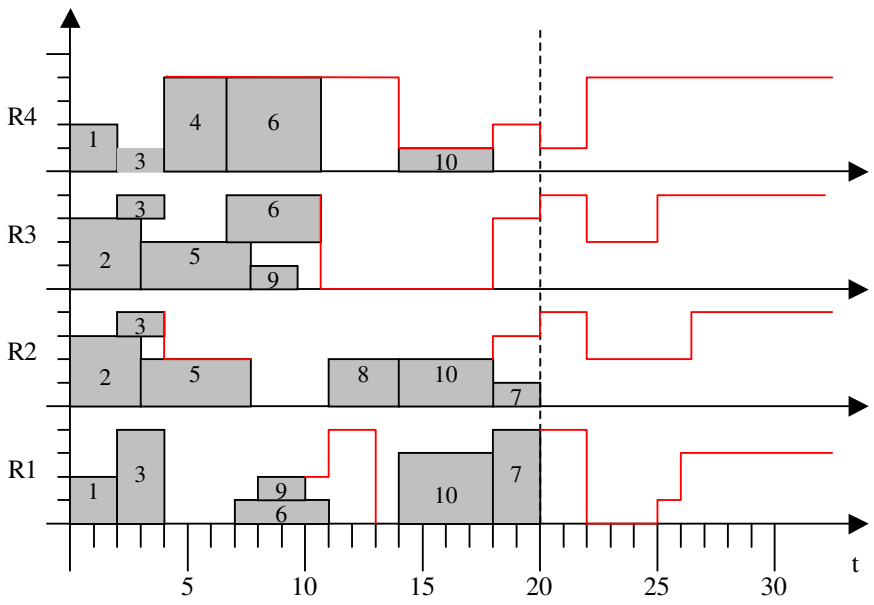
$$c_{b_i} = s_{b_i} + p_{b_i}$$

3) Jei $i = n$, tai $\{T(s) = c_{b_n} + p_{n+1}$, baigti$\}$,

Kitaip $\{j = j+1$, eiti į 2$)\}$.

Nesunku pastebėti, kad algoritmo žingsniui 2) reikia atlikti $O(n \cdot m)$ elementarių operacijų. Tad dekodavimo procedūros sudėtingumas laiko prasme yra $O(n^2 \cdot m)$.

Sudarant tvarkaraštį gali pasitaikyti, jog nuoseklumo sąryšiai leis vykdyti kai kurias užduotis, bet tam prieštaraus išteklių ribojimai. Tuomet užduočių atlikimą tenka atidėti, kol atsiras reikalingi ištekliai. Tvarkaraščio sudarymo uždavinys žymiai pasunkėja, kai išteklių apimtys kinta projekto vykdymo metu. 6.1 pav. pateiktas tvarkaraštis, sudarytas nuosekliu dekoderiu remiantis 3.3 pav. duomenimis. Šis tvarkaraštis iliustruoja bendresnį atvejį, kai turimų išteklių apimtys laikui bėgant kinta.



6.1 pav. Užduočių tvarkaraščio ir išteklių naudojimo grafinis atvaizdavimas laiko skalėje

Pasinaudodami užduočių pirmumo sąrašo nuoseklaus dekodavimo procedūra, galime konstruoti tvarkaraščių optimizavimo algoritmus, pritaikydami įvairias euristines procedūras. Paprasčiausia būtų pritaikyti Monte Karlo metodą atsitiktinai generuojant tam tikrą skaičių pirmumo sąrašų ir atrenkant iš jų leistiną sąrašą su minimalia projekto atlikimo trukme. Įvedus temperatūros bei aplinkos reguliavimo funkcijas, tvarkaraščiams optimizuoti galime pritaikyti modeliuojamojo atkaitinimo algoritmą. Taip pat galima sukonstruoti genetinį tvarkaraščių optimizavimo algoritmą, kiekvieną sąrašą laikydami atskira chromosoma ir kurdami bei modifikuodami tokių chromosomų populiaciją.

6.4 Tvarkaraščių paieška Monte-Karlo metodu

Atsitiktinė paieška yra paprasčiausias tvarkaraščio sudarymo būdas. Pateiksime tokios paieškos efektyvumo tyrimo Monte Karlo metodu algoritmą.

Žingsnis 1. $K = 0$, $M = 0$, $SUMZ = 0$, generuojamas atsitiktinis pirmumo sąrašas b', $b' = (b_1, ..., b_n)$, n – užduočių skaičius $Z^* = TF(b')$. Jo leistinumas yra nustatomas pagal pilnąją sąryšių matricą (žiūr. 6.2 skyrelį). Jei sprendinys nėra leistinas, tol atsitiktinai generuojame pirmumo sąrašus, kol gaunamas leistinas sąrašas.

Žingsnis 2. $K = K + 1$. Generuojame leistiną pirmumo sąrašą b.

Žingsnis 3. Skaičiuojame tikslo funkcijos reikšmę $Z = TF(b)$. Jei $Z < Z^*$, tai $Z^* = Z$.

Žingsnis 4. Jei $K < MAXIT$, tai eiti į **Žingsnis 2**, kitaip į **Žingsnis 5**.

Žingsnis 5. $SUMZ = SUMZ + Z^*$. $M = M + 1$. $K = 0$.

Generuojame naują atsitiktinį leistiną pirmumo sąrašą b'. $Z^* = TF(b')$.

Žingsnis 6. Jei $M < MAXD$, tai eiti į Žingsnis 2, kitaip – į Žingsnis 7.

Žingsnis 7. $SUMZ = SUMZ / MAXD$.

Atsitiktinė paieška yra naudojama palyginimui su kitais tvarkaraščio optimizavimo metodais. Yra prasminga nagrinėti tik tokius algoritmus, kurie yra efektyvesni už atsitiktinę paiešką.

6.5 Tolydusis optimizavimas modeliuojamojo atkaitinimo metodu

Modeliuojamojo atkaitinimo metodo taikymas tvarkaraščiams optimizuoti buvo nagrinėjamas darbuose (Bouleimen, Lecocq, 1998, Bouleimen, Lecocq, 2003, Kim, Moon, 2003). Reikia pastebėti, kad pritaikant šį algoritmą yra realizuojamos ne visos jo paradigmos dalys, kurios yra aprašytos 5.2 skyrelyje. Pavyzdžiui, daugelis autorių savo algoritmus vadina modeliuojamojo atkaitinimo, jeigu juose realizuotas tikimybinis Metropolio sprendinių patvirtinimo kriterijus. Modeliuojamojo atkaitinimo algoritmai su kintamos aplinkos gyliu dar nėra realizuoti tvarkaraščių su ribojimais uždaviniams. Šiame darbe išnagrinėsime modeliuojamojo atkaitinimo algoritmo schemą, aprašytą 5.2 skyrelyje ir pritaikysime ją tvarkaraščiams su ribojimais optimizuoti. Ši schema teoriškai buvo išnagrinėta tolydaus optimizavimo atveju (Yang, 2000), kur buvo nustatytos

algoritmo konvergavimo sąlygos. Todėl panagrinėsime šios schemos savybes tolydaus optimizavimo atveju, po to perkeldami jas tvarkaraščių optimizavimo algoritmams.

Nagrinėkime optimizavimo uždavinį (minimizavimo)

$$\min_{x \in \Xi} f(x), \tag{6.1}$$

čia Ξ yra R^n poaibis, f yra reali funkcija, apibrėžta srityje Ξ ir turinti globalųjį minimumą f^* srityje Ξ.

Aprašysime MA algoritmą tolydaus optimizavimo atveju sprendžiant (6.1) uždavinį.

1 žingsnis. Pasirenkame pradinį tašką $x^o \in \Xi$, pradinę temperatūrą $T_o > 0$, tam tikrą nuo temperatūros priklausančią generavimo tikimybinę tankio funkciją, atitinkamą temperatūros atnaujinimo funkciją, ir seką $\{\rho_j; j \geq 0\}$ monotoniškai mažėjančių teigiamų skaičių. Suskaičiuojame $f(x^o)$. Nustatome $X^o = x^o$ ir $k = 0$.

2 žingsnis. Generuojame atsitiktinį vektorių Z^k naudodami generavimo tikimybinę tankio funkciją. Jei egzistuoja $1 \leq i \leq n$ toks, kad $|Z_i^k| < \rho_k$, kur Z_i^k yra *i*-asis vektoriaus Z^k komponentas, kartoti **2 žingsnį**. Kitu atveju, gaunamas naujas taškas Y^k pridedant atsitiktinį vektorių Z^k prie dabartinės iteracijos taško X^k,

$$Y^k = X^k + Z^k \tag{6.2}$$

Jei $Y^k \notin \Xi$, kartojame **2 žingsnį**; kitu atveju, skaičiuojame $f(Y^k)$.

3 žingsnis. Naudojame Metropolio **kriterijų** naujam iteracijos taškui X^{k+1} parinkti, t.y.

generuojame atsitiktinį skaičių η tolygiai pasiskirsčiusį intervale (0,1) ir tikimybę $P(Y^k, X^k, T^k)$. Jei $\eta \leq P(Y^k, X^k, T^k)$, tada $X^{k+1} = Y^k$ ir $f(X^{k+1}) = f(Y^k)$. Kitu atveju – paliekame $X^{k+1} = X^k$ ir $f(X^{k+1}) = f(X^k)$.

$$P(Y^k, X^k, T^k) = \min\{1, \exp\{[f(X^k) - f(Y^k)] / T_k\}\}. \tag{6.3}$$

4 žingsnis. Jei iš anksto numatyta sustojimo sąlyga (pvz. nurodytas iteracijų skaičius) yra patenkinta, tada sustoti. Kitu atveju atnaujiname temperatūrą naudodami temperatūros atnaujinimo funkciją ir grįžtame prie **2 žingsnio**.

Šio algoritmo konvergavimo sąlygos nustato temperatūros atnaujinimo funkcijos pavidalą, parametrus bei teigiamų skaičių sekos $\{\rho_j; j \geq 0\}$ parinkimą, garantuojantį modeliuojamojo atkaitinimo algoritmo konvergavimą į tikslo funkcijos globalųjį minimumą tiriamoje srityje. Teigiamų skaičių seka $\{\rho_j; j \geq 0\}$ kiekvienoje iteracijoje nustato mažiausią ribą, kurią turi viršyti generuojamo vektoriaus Z^k koordinatės. Yang nustatė, kad temperatūros atnaujinimo funkcija T_k bei teigiamų skaičių seka $\{\rho_j; j \geq 0\}$ turi būti suderintos su generavimo tikimybine tankio funkcija. Straipsnyje išnagrinėti keli generavimo skirstiniai**,** pasižymintys Pareto savybe (Yang, 2000). 6.1

lentelėje yra pateikiami du generavimo skirstiniai ir juos atitinkančių algoritmo parametrų kitimo priklausomybės.

6.1 lentelė. Generavimo skirstiniai ir atitinkamos MA algoritmo charakteristikos

Tankis	Tikimybė F(z)=P(Z<z)	ρ_k	T_k
(a) $p(z,T_k) = T_k / \pi(z^2 + T_k^2)$	$P(Z < z,T_k) = \frac{1}{\pi} \cdot (arctg(\frac{z}{T_k}) + \frac{\pi}{2})$	$\rho_k = \rho_0 / k^{\gamma/4n}$	$T_k = T_0 / k^{1/n}$
(b) $p(z,T_k) = T_k^{1/m} / 2m[\lvert z\rvert + T_k]^{(m+1)/m}$	$P(Z < z,T_k) = \frac{1}{2 \cdot \left(1 - \frac{z}{T_k}\right)^{1/m}}, \quad z < 0,$ $P(Z < z,T_k) = 1 - \frac{1}{2 \cdot \left(\frac{z}{T_k} + 1\right)^{1/m}}, \quad z > 0$	$\rho_k = \rho_0 / k^{\gamma m/\lambda(m+1)}$	$T_k = T_0 / k^{m/n}$

Remiantis 6.1 lentele, buvo gautos taisyklės dyžiams Z_k antrajame algoritmo žingsnyje modeliuoti.

Globaliojo optimizavimo algoritmų ir metodų efektyvumas yra tiriamas naudojant testines funkcijas, kurių optimumo taškai iš anksto žinomi (Schoen, 1993). Nagrinėjamam algoritmui testuoti buvo naudojamos šios testinės funkcijos: Branino (Branin, turi 3 globaliuosius optimumus), Rastrigino (Rastrigin, maždaug 50 lokaliųjų ekstremumų ir vienas globalusis), Goldšteino-Praiso (Goldstein-Price, G-P, 4 lokalieji ekstremumai ir vienas globalusis) (Žilinskas, 1986).

Buvo tiriamos tokios algoritmo charakteristikos: minimizuojamos funkcijos reikšmė bei globaliojo optimumo radimo tikimybė po tam tikro iteracijų skaičiaus. Šios charakteristikos buvo vertinamos Monte-Karlo metodu, atliekant N nusileidimų po K iteracijų ($N = 1000$, kai $K <= 3000$, $N = 500$, kai $3000 < K <= 10000$ ir $N = 100$, kai $K > 10000$). Efektyvumo charakteristikų pasikliautinieji intervalai buvo nustatomi remiantis normaline aproksimacija. Globaliojo optimumo radimo tikimybė buvo vertinama apskaičiuojant optimizavimo taško patekimo dažnumą į globaliojo optimumo aplinką: $|f_k - f^*| < \varepsilon$. Pradinis taškas buvo parenkamas atsitiktinai.

Modeliuojant MA algoritmą kelioms testinėms funkcijoms su dviem skirtingais skirstiniais bei manipuliuojant įvairiais leidžiamais keisti parametrais buvo siekiama išsiaiškinti:

- kuris iš duotų skirstinių užtikrina spartesnį konvergavimą į globalųjį minimumą pagal tikslo funkcijos reikšmę;
- kokios yra „nusileidimo“ į globalųjį minimumą tikimybės ir kaip jas gali paveikti tam tikrų parametrų keitimas;
- koks reikalingas iteracijų skaičius, kad surasti globalųjį minimumą su reikalinga tikimybe.

6.2 lentelė. Konvergavimas į globalųjį minimumą pagal funkcijos reikšmę F_k

Iteracijų skaičius	F_k reikšmės					
	Branino f-ja		Goldšteino-Praiso f-ja		Rastrigino f-ja	
	(a) skirst.	(b) skirst.	(a) skirst.	(b) skirst.	(a) skirst.	(b) skirst.
100	1,41265	0,52039	13,45722	9,87812	-0,40869	-1,67522
500	0,83006	0,41728	3,75233	5,53644	-1,20333	-1,94236
1000	0,68788	0,40682	3,36933	4,73926	-1,32400	-1,98219
3000	0,54042	0,40129	3,16755	3,78732	-1,64010	-1,99647
10000	0,48771	0,39878	3,10922	3,00125	-1,79196	-1,99814
30000	0,44670	0,39828	3,05393	3,00038	-1,90625	-1,99957
Glob.min.	**0,397887**		**3,00**		**-2,00**	

6.3 lentelė. Patekimo į globalųjį minimumą tikimybės G-P funkcijai ($\varepsilon = 0,01$)

Iteracijų skaičius	Patekimo į globalųjį minimumą tikimybės		
	m=2	**m=3**	**m=4**
100	**0,025**±0,008	**0,160**±0,019	**0,248**±0,023
200	**0,145**±0,018	**0,482**±0,026	**0,427**±0,026
300	**0,261**±0,023	**0,598**±0,026	**0,495**±0,026
500	**0,485**±0,026	**0,627**±0,025	**0,527**±0,026
1000	**0,764**±0,022	**0,658**±0,025	**0,530**±0,026

Šio skaičiuojamojo eksperimento rezultatai pateikti 6.2 ir 6.3 lentelėse. Remiantis šiais rezultatais galima tvirtinti, kad Pareto tipo modelio parametro $\alpha = 1/m$ reguliavimas leidžia padidinti globaliojo optimumo suradimo tikimybę. Tad tokių modelių taikymas suteikia galimybę pagerinti euristinių algoritmų efektyvumą.

Algoritmo efektyvumui padidinti, kol iteracijų skaičius yra mažas, reiktų parinkti mažą parametro α reikšmę, kuri vėliau, didėjant iteracijų skaičiui, turėtų būti didinama (Felinskas, Sakalauskas, 2003).

6.6 Tvarkaraščio optimizavimas modeliuojamojo atkaitinimo ir kintamos aplinkos metodu

Panagrinėkime plačiau modeliuojamojo atkaitinimo algoritmą (MA), besiremiantį pirmumo sąrašu bei nuoseklaus dekodavimo procedūra. MA tipo algoritmuose atsitiktiniu būdu yra generuojami sprendiniai ir remiantis tam tikromis taisyklėmis yra pereinama nuo vieno sprendinio prie kito. Nauji sprendiniai yra generuojami pasirinkto sprendinio aplinkoje, apskaičiuojant tikslo funkcijos reikšmes. Paprastai aplinkos gylis (spindulys) yra mažinamas optimizavimo metu, pradedant nuo tam tikro fiksuoto dydžio. Perėjimui prie naujojo sprendinio yra taikoma Metropolio taisyklė, leidžianti su tam tikra tikimybe pasirinkti sprendinį su blogesne tikslo funkcijos reikšme. Atitinkamai parinkus generavimo, aplinkos mažinimo bei Metropolio taisyklės parametrus, galima pasiekti algoritmo konvergavimą į globalųjį optimumą (Yang, 2000). MA algoritmo parametrų reguliavimas konvergavimo greičiui bei tikslumui pagerinti buvo tiriamas darbe (Felinskas, Sakalauskas, 2003).

Konstruodami MA algoritmą tvarkaraščiams optimizuoti, du pirmumo sąrašus laikysime gretimais, jeigu jie gali būti gauti vienas iš kito pritaikius vieną elementarią operaciją (žiūr. 3.9.2 skyrelį). Elementaria operacija laikysime užduoties perkėlimo (Shift) operaciją, kai kuri nors užduotis yra perkeliama į kitą sąrašo vietą, ir užduočių sukeitimo (swap) operaciją, kai kurios nors dvi užduotys sąraše yra sukeičiamos vietomis.

Panagrinėsime RITSU sprendimo MA algoritmą.

Tegul l ir q yra dvi pirmumo sąrašo pozicijos, $l \neq q$, $1 \le l,q \le n$.

Pirmumo sąrašas b' iš sąrašo b po perkėlimo (shift) operacijos gaunamas, atitinkamai pernumeruojant pirmumo sąrašą: jei $q > l$, tai $b'_x = b_{x+1}$, $l \le x < q$, jei $q < l$, tai $b'_x = b_{x-1}$, $q < x \le l$. Po to priskiriame $b'_q = b_l$. Kiti b' elementai sutampa su b elementais.

Pirmumo sąrašas b' iš sąrašo b po sukeitimo (swap) operacijos gaunamas taip: $b'_q = b_l$, $b'_l = b_q$. Kiti b' elementai sutampa su b elementais.

Algoritmo parametrams reguliuoti yra taikomos taisyklės

$\rho_k = \rho_0 / k^{\alpha}$, $t_k = t_0 / k^{\beta}$, $0 < \alpha,\beta < 1$

Tokio tipo taisyklių įtaka konvergavimo greičiui ir tikslumui buvo aptarta darbe (Yang, 2000).

Užrašykime tokį MA algoritmą:

1) $k = 0$.

2) Tegul yra atlikta k algoritmo žingsnių. Tarkime, turime pirmumo sąrašą b, kuriame tikslo funkcijos reikšmė yra $Z1 = T(b)$. Nustatome žingsnio parametrus ρ_k ir t_k.

3) Atsitiktinai generuojame skaičius q ir l, $l \neq q$, $1 \le l,q \le n$. Su tikimybe $p = 0{,}5$ atliekame arba perkėlimo operaciją, arba užduočių sukeitimo operaciją.

4) Jei tokiu būdu gautas pirmumo sąrašas nėra leistinas, tai kartojame žingsnį 3), kol gauname leistinąjį sąrašą.

5) žingsnius 3) ir 4) kartojame ρ_k kartų. Sąrašą, gautą atlikus ρ_k leistinųjų elementarių operacijų, pažymėkime b'.

6) Skaičiuojame tikslo funkciją $Z2 = T(b')$ gautam pirmumo sąrašui, pritaikydami nuoseklaus dekodavimo procedūrą.

7) pirmumo sąrašas keičiamas pagal Metropolio taisyklę:

jei $\eta < exp((Z1-Z2) / t_k)$, tai $b' = b$. $K = k + 1$. Jei $k < k_{max}$, tai kartojame žingsnį 2).

Programinė įranga, realizuojanti sukurtąjį modeliuojamojo atkaitinimo algoritmą, yra aprašyta priede E.

Sukurtojo modeliuojamojo atkaitinimo algoritmo efektyvumo eksperimentinio tyrimo rezultatai yra pateikiami 7.2, 7.4 skyreliuose.

6.7 Tvarkaraščio optimizavimo genetinio algoritmo sudarymas, remiantis užduočių pirmumo sąrašu

Modeliuojamojo atkaitinimo metodas priklauso nuoseklios paieškos metaeuristikoms. Genetinių algoritmų nagrinėjimas leidžia ištirti evoliucinės paieškos metaeuristikų realizavimo aspektu, taikant užduočių pirmumo sąrašą. Nagrinėsime tokį genetinį algoritmą.

Žingsnis 1. Pradinės populiacijos generavimas. Generuojame M leistinų pirmumo sąrašų (M = 40, 20, ...). Sukonstruojame matricą $BP = (b_{ij})$, $1 \le i \le M$, $1 \le j \le N$, N – užduočių skaičius, M – sąrašų populiacijoje skaičius.

$K = 1$. $TF^{best} = TF$(*bet kuris pirmumo sąrašas*).

Žingsnis 2. Kryžminimas (Crossover). Kiekvienoje K-ojoje iteracijoje atliekama pirmumo sąrašų kryžminimo procedūra, kuri atliekama su kiekviena pirmumo sąrašų pora.

Tegul b' ir b^* yra 2 pirmumo sąrašai („tėvai“). Generuojama atsitiktinė pozicija w.

Iš dviejų leistinų sąrašų $(b'_1,...,b'_w,b'_{w+1},...,b'_N)$ ir $(b^*_1,...,b^*_w,b^*_{w+1},...,b^*_N)$ gaunami 2 nauji pirmumo sąrašai b'' ir b^{**}. $b'' = (b'_1,...,b'_w,b^*_{k1},...,b^*_{k(N-w)})$, čia $b^*_{k1},...,b^*_{k(N-w)}$ – likę vektoriaus b^* komponentai, išlaikant jų tarpusavio eiliškumą, eliminavus (išbraukus) paeiliui pirmuosius w vektoriaus b' komponentus, t.y. $(b'_1,...,b'_w)$. Analogiškai sudaromas $b^{**} = (b^*_1,...,b^*_w,b'_{k1},...,b'_{k(N-w)})$.

Reikia pastebėti, jog naujai gauti pirmumo sąrašai b' ir b^* taip pat yra leistini, nes išlaikomi visi buvę nuoseklumo sąryšiai. Šią kryžminimo procedūrą atliekame su visomis matricos BP sąrašų poromis (M / 2 porų). $BP \xrightarrow{crossover} BC$.

Žingsnis 3. Mutacija (Mutation). Kiekvienoje K-ojoje iteracijoje atliekame mutacijos procedūrą (taip teoriškai suteikiama galimybė gauti bet kokį pirmumo sąrašą iš visos sprendinių erdvės). Iš anksto nustatyta mutacijos tikimybė pp nurodo, kurioms chromosomoms (pirmumo sąrašams) taikyti mutacijos procedūrą. Ši procedūra atliekama su nustatytais pirmumo sąrašais (matricos BC eilutėmis). Sugeneruojami atsitiktiniai skaičiai l ir q, $l \ne q$, $1 \le l,q \le N$. Su tikimybe $p = 0{,}5$ q-ąją užduotį perkeliame prieš l-tąją užduotį arba q-ąją ir l-tąją užduotis sukeičiame vietomis. Jei gautasis sąrašas nėra leistinas, šį keitimą atšaukiame. $BC \xrightarrow{mutation} BC^{MUT}$.

Žingsnis 4. Atranka (Selection). Iš $M+M$ pirmumo sąrašų („tėvų“ ir „vaikų“) reikia vėl atrinkti M pirmumo sąrašų. (galima atrinkti ir palikti tik tuos pirmumo sąrašus, su kuriais tikslo funkcija $Z = TF(b)$ įgyja geriausias reikšmes. Atrankos metodai detaliai aprašyti (Glibovec, Medvidj, 2003).

Sujungus matricas BP ir BC, gauname naują matricą BPC, kurios matmenys $2M*N$.

Imame 2 atsitiktinius sąrašus b' ir b^* iš matricos BPC. Tegul $Z1 = TF(b')$, $Z2 = TF(b^*)$. Jei $Z1 < Z2$, tada sąrašas b' paliekamas, b^* naikinamas. Priešingu atveju, kai $Z1 \ge Z2$, b^* paliekamas, o b' naikinamas. Po M dvikovų vėl lieka $M*N$ matmenų matrica.

Žingsnis 5. Randama geriausia pasiekta tikslo funkcijos reikšmė $TF* = \min\limits_{1 \le i \le M} TF(b^i)$.

Jei $TF^{*} < TF^{best}$, tada $TF^{best} = TF^{*}$.

Žingsnis 6. Jei $K = K_{max}$, tada stabdome algoritmą, kitaip $K = K + 1$ ir einame į **Žingsnį 2**.

Programinė įranga, realizuojanti sukurtąjį genetinį algoritmą, yra aprašyta priede E.

Sukurtojo genetinio algoritmo efektyvumo eksperimentinio tyrimo rezultatai yra pateikiami 7.3-7.4 skyreliuose.

6.8. Išvados

1. Nagrinėti ribotų išteklių tvarkaraščių uždavinių sudarymo ir optimizavimo algoritmai, besiremiantys tvarkaraščio kodavimu užduočių pirmumo sąrašu ir nuoseklaus dekodavimo procedūra. Toks tvarkaraščio kodavimas leidžia taikyti įvairius euristinius optimizavimo metodus, galima lengviau generuoti naujus sprendinius iš leistinos sprendinių erdvės, pritaikant elementarias operacijas.
2. Sudarytas MA algoritmas RITSU uždaviniams spręsti, kuriame realizuotas aplinkos gylio reguliavimas pagal Pareto dėsnį, suderintas su temperatūros reguliavimu bei atsitiktinių sprendinių generavimu.
3. Sukurtas genetinis algoritmas RITSU optimizuoti, sukuriant kryžminimo, mutacijos ir atrankos procedūras, besiremiančias užduočių pirmumo sąrašais.

7 skyrius. Euristinių tvarkaraščių optimizavimo metodų eksperimentinis tyrimas

Šiame skyriuje panagrinėsime tvarkaraščių optimizavimo euristiniais metodais eksperimentinio tyrimo metodologiją, euristinių tvarkaraščių optimizavimo algoritmų testavimo ir praktinio taikymo rezultatus.

Kadangi dauguma tvarkaraščių uždavinių yra NP-pilnieji, todėl optimalius tokių uždavinių sprendinius per priimtiną laiką galime rasti tik mažos apimties uždaviniams. Testuojant sukurtus algoritmus bei tiriant jų efektyvumą yra pasinaudojama standartinių testinių pavyzdžių duomenų bazėmis, ir apie sukurtų algoritmų privalumus yra sprendžiama lyginant didelio skaičiaus testinių uždavinių sprendimo rezultatus. Šiame skyriuje nagrinėsime plačiai taikomą biblioteką PSPLib bei pateiksime kompiuterinio tyrimo rezultatus, taikant euristinius algoritmus šios bibliotekos uždavinių rinkiniams spręsti.

Remiantis testavimo rezultatais yra pateikiamos rekomendacijos praktiniam euristinių algoritmų realizavimui bei taikymui. Pateikiami HP montažinių plokščių surinkimo uždavinio sprendimo rezultatai.

7.1 Testinių uždavinių duomenų bazė

Testinių pavyzdžių rinkinių iš standartinių uždavinių duomenų bazių sprendimo rezultatų palyginimas yra plačiai naudojamas būdas sukurtų algoritmų efektyvumui įvertinti. Yra siekiama, kad sukurti algoritmai pagerintų geriausius žinomus testinių uždavinių sprendinius arba pasiektų bent tą patį rezultatą su mažesnėmis skaičiuojamosiomis sąnaudomis. Yra žinomos įvairios tokios duomenų bazės. Žinomiausia RITSU testinių pavyzdžių duomenų bazė yra Projektų Tvarkarščių Problemų Biblioteka (PSPLib, Project Scheduling Problem Library). Aptarsime darbą su šios internetinės bibliotekos pavyzdžiais.

7.1.1. Bibliotekos PSPLib aprašymas

Bibliotekoje PSPLib (PSPLIB – A project scheduling library, 1996**,** Kolisch, Sprecher, 1996) galima rasti skirtingų uždavinių rinkinių įvairaus tipo tvarkaraščių su ribotais ištekliais sudarymo uždaviniams spręsti. Kartu su uždavinio formalizuotu aprašymu yra pateikiami uždavinio sprendiniai, gauti įvairių autorių įvairiais algoritmais. Duomenų rinkiniai gali būti naudojami sprendimo procedūroms tikrinti. Pavyzdžiai (atskiri atvejai) yra sugeneruoti naudojant standartinį projektų generatorių ProGen. Tyrinėtojai gali atsisiųsti testų rinkinius savo algoritmams tikrinti, taip pat nusiųsti savo rezultatus, kurie gali būti patalpinti šioje bibliotekoje. Daugiau informacijos apie biblioteką, uždavinių generavimo detales, eksperimentus, modelių tipus, uždavinių parametrus galima rasti (Kolisch, Sprecher, 1996). Pagrindiniai parametrai pateikti tolesniuose skyreliuose.

7.1.2 Uždavinių duomenų ir duomenų apie sprendinius gavimas

Atskirų pavyzdžių archyvuoti failai yra užrašomi formatu *x.mm.tgz*, kuriame yra pilnas uždavinio atvejų rinkinys, čia *x* yra pavyzdžio identifikatorius (gali būti užrašytas keliais simboliais), *mm* žymi pavyzdžio užduočių būsenų skaičių: *sm* – single-mode, *mm* – multi-mode. Pavyzdžiuose, žymimuose *sm*, darbas gali būti tik vienoje vykdymo būsenoje, o pavyzdžiuose, žymimuose *mm*, vykdymo metu darbas gali pereiti keletą būsenų. Pavyzdžiui, vienos būsenos (single-mode) pilnas rinkinys su 480 uždavinių atvejų 30-iai užduočių yra pateiktas *j30.sm.tgz*. Bibliotekos archyve yra failai, kuriuose pateikiami žinomi šių uždavinių tikslūs sprendiniai (optimal), o taip pat geriausi apytikriai sprendiniai, pasiekti euristiniais algoritmais (heuristic), jeigu tikslus uždavinio sprendinys nėra žinomas. Jei tikslus uždavinio sprendinys nėra žinomas, yra pateikiamos apatinės optimumo ribos (lover-bounds). Pavyzdžiai yra surūšiuoti kaip testų atskiri atvejai pagal būsenų skaičių (vienos būsenos, daugelio būsenų) ir pagal užduočių skaičių (30, 60 ir t.t.). Minėtų sprendinių formatų detalius aprašus rasite (PSPLIB – A project scheduling library, 1996).

7.1.3 Parametrų nustatymai

Bibliotekoje yra labai daug pavyzdžių RITSU spręsti, t.y. po 480 pavyzdžių kiekvienam uždaviniui su $n = 30$, $n = 60$ ir $n = 90$ užduočių ir po 600 pavyzdžių su $n = 120$ užduočių. Išteklių tipų skaičius yra 4. Visi pavyzdžiai padalinti į klases, po 10 pavyzdžių kiekvienai klasei, sugeneruotų atsitiktinai, naudojant trijų parametrų fiksuotas reikšmes:

- $NC \in \{1,5;\ 1,8;\ 2,1\}$ – vidutinis skaičius užduočių, kurios turi būti atliktos prieš kiekvieną užduotį.
- $RF \in \{1;2;3;4\}$ – išteklių tipų skaičius, reikalingas kiekvienai užduočiai atlikti.
- $RS \in \{0,2;\ 0,5;\ 0,7;\ 1,0\}$ – duotų išteklių kiekis kiekvienu laiko momentu. Reikšmė $RS = 0,2$ atitinka pavyzdžius su minimaliu kiekiu duotų išteklių, kurių pakanka uždaviniui išspręsti. Reikšmė $RS = 1,0$ atitinka atvejus be išteklių ribojimų.

Yra žinoma, kad parametrų reikšmės $RF = 4$, $RS = 0,2$ atitinka sunkių uždavinių klasę. Jie yra patalpinti failuose su vardo pradžia *j3013, 13029, j3045*, kai užduočių skaičius $n = 30$, *j6013, j6029, j6045*, kai užduočių skaičius $n = 60$, ir *j12016, j12036, j12056*, kai užduočių skaičius $n = 120$, atitinkamai su parametro reikšmėmis $NC = 1,5;\ 1,8;\ 2,1$.

7.1.4 Testų charakteristikos

Šiuo metu galima rasti 2 testų rinkinius vienos būsenos (single-mode RCPSP) uždaviniams spręsti ir 25 testų rinkinius daugelio būsenų (multi-mode RCPSP) uždaviniams. Pradiniai parametrai sugrupuoti į tris klases. *Pirma klasė* – tai fiksuoti (fixed) parametrai, kurie yra konstantos visiems testų rinkiniams. *Antra klasė* – pagrindiniai (base) parametrai iš kurių iš esmės

vienas yra suderintas būtent su kiekvienu konkrečiu testų rinkiniu. *Trečia klasė* – kintami (variable) parametrai, kurie sistemingai kinta kiekviename testų rinkinyje.

Toliau lentelėse pateikti parametrų nustatymai dažniausiai sprendžiamiems vienos būsenos (single-mode RCPSP) uždaviniams. Visus kitus detalius nustatymus kitiems uždaviniams rasite (PSPLIB – A project scheduling library, 1996).

7.1 lentelė. Fiksuotų parametrų nustatymai (SMRCPSP ir MMRCPSP)

P_1^R	P_2^R	P_1^N	P_2^N	ε_{NET}	ε_{RF}
0,00	*1,00*	*0,00*	*1,00*	*0,05*	*0,05*

7.2 lentelė. Pagrindinių parametrų nustatymai (SMRCPSP)

	J	M_j	d_j	$\|R\|$	U_R	Q_R	$\|N\|$	U_N	Q_N	S_1	S_j	P_J	P_j
min	*30*	*1*	*1*	*4*	*1*	*1*	*0*	*0*	*0*	*3*	*1*	*3*	*1*
max	*30*	*1*	*10*	*4*	*10*	*2*	*0*	*0*	*0*	*3*	*3*	*3*	*3*

7.3 lentelė. Kintamų parametrų nustatymai (SMRCPSP)

Parametras	*Lygiai*			
NC	*1,50*	*1,80*	*2,10*	
RF_R	*0,25*	*0,50*	*0,75*	*1,00*
RS_R	*0,20*	*0,50*	*0,70*	*1,00*

Vienos būsenos RITSU uždaviniai yra sugeneruoti naudojant 7.1-7.3 lentelėse duotus fiksuotus, pagrindinius ir kintamus parametrų nustatymus. Suskaičiavus galimas kombinacijas naudojant parametrus *NC,* RF_R, RS_R, kiekvienai generuojant po 10 uždavinių, iš viso gauname 3x4x4x10=480 testinių uždavinių kiekvienam rinkiniui.

7.4 lentelė. Atskiri vienos būsenos RITSU atvejai

Atskirų atvejų rinkinys	*P*	*E*	*Tipas*	*Keičiama pagrindinių parametrų 7.2 lentelė*	*Kintamųjų nustatymai*	*atskirų atvejų skaičius*	*sprendinys gautas pagal*
J30	*[1..48]*	*[1..10]*	*SM*	$J^{min}= J^{max}=30$	*7.3 lentelė*	*480*	*opt.*
J60	*[1..48]*	*[1..10]*	*SM*	$J^{min}= J^{max}=60$	*7.3 lentelė*	*480*	*hrs.*

7.4 lentelėje pateiktas atskirų uždavinių atvejų aprašymas. 1-ajame stulpelyje (atskirų atvejų rinkinys) yra bibliotekoje saugomų atskirų atvejų failų vardų prefiksas (pavadinimo pradžia). 2-asis stulpelis (*P*) rodo galimų kombinacijų indeksų intervalą, atspindintį kintamų parametrų derinius. 3-

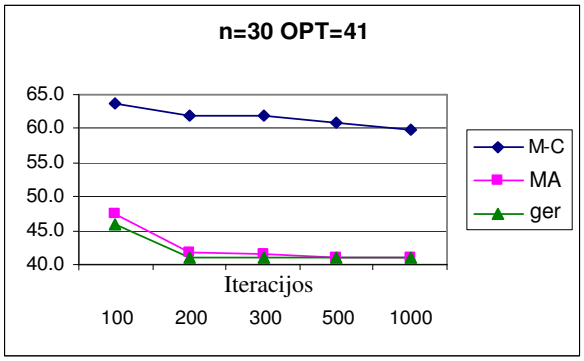
iasis stulpelis (*E*) nurodo galimų atskirų atvejų indeksų intervalą. 4-ajame stulpelyje pateiktas akronimo SMRCPSP santrumpa *SM*. Ji naudojama failo prievardyje. 5-asis stulpelis rodo keičiamus pagrindinius parametrus, kai užduočių skaičių reikia keisti į 30 ir į 60 atitinkamai. 6-asis stulpelis nurodo į lentelę su kintamais parametrų reikšmių lygiais. 7-asis stulpelis rodo visų atskirų atvejų skaičių testiniame rinkinyje. 8-asis stulpelis rodo, kaip yra gautas testinių rinkinių sprendinys (opt. – tikslaus sprendinio radimo metodu, hrs. – euristiniu metodu).

Pilnas failo vardas, pavyzdžiui, J301210.sm rodo į atskirų atvejų rinkinį J30 (30 užduočių), kintamų parametrų kombinaciją 12 ir uždavinio numerį 10 vienos būsenos (single-mode) RCPSP uždavinių. Kintamų parametrų nustatymų lygį 7.4 lentelėje pateiktiems atvejams rasite atitinkamai failuose J30PAR.SM ir J60PAR.SM. Optimalios tikslo funkcijos reikšmės atskiriems J30 testinių rinkinių atvejams yra gautos ir pateiktos J30OPT.SM. Pradedant nuo 60 užduočių, bendru atveju uždaviniai negali būti išspręsti naudojant tikslaus sprendinio radimo procedūras. Tam reikalingi euristiniai metodai.

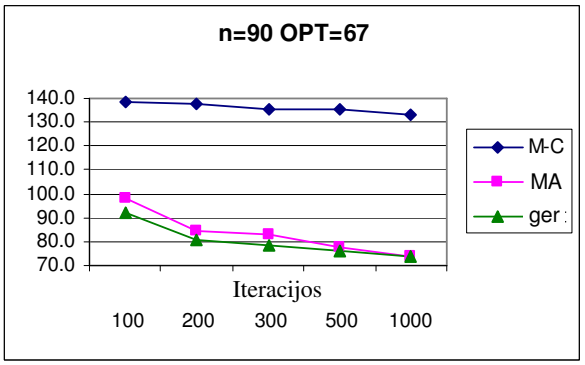
7.2 RITSU sprendimo MA algoritmu efektyvumo tyrimo rezultatai

Sukonstruotas MA algoritmas buvo testuojamas naudojant duomenų rinkinius iš tvarkaraščių uždavinių bibbliotekos PSPLib. Buvo paimta keletas testinių duomenų rinkinių (pateikiami ir geriausi žinomi sprendiniai, o 30 užduočių atveju – globalusis optimumas) su įvairiu užduočių skaičiumi, jų nuoseklumo sąryšiais, įvairiais resursų poreikiais ir ištekliais. Buvo palygintas tikslo funkcijos konvergavimo greitis, naudojant atsitiktinę paiešką (pažymėta „M-C“) bei modeliuojamojo atkaitinimo metodą (žiūr. 7.1-7.4 pav.– atitinkamai su 30, 60, 90 ir 120 užduočių). Atsitiktinės paieškos metodu buvo atlikta po 100 nusileidimų bei fiksuojamas pasiektas tikslo funkcijos reikšmės vidurkis po 100, 200, 300, 500 ir 1000 iteracijų. Kiekvienoje iteracijoje generuojamas leistinasis pirmumo sąrašas, ieškoma tikslo funkcijos reikšmės „pagerėjimo“ lyginant su ankstesne reikšme.

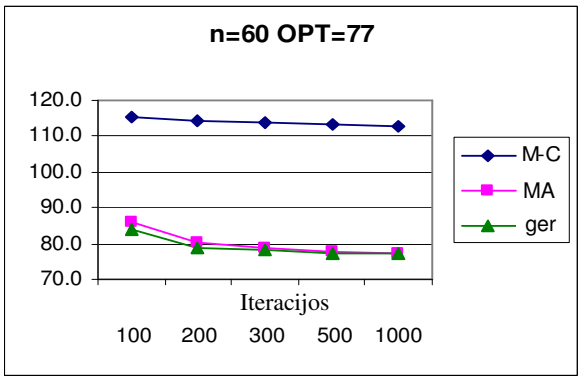
Tokie pat lyginamieji tyrimai buvo atlikti taikant modeliuojamojo atkaitinimo metodą (paveikslėliuose pažymėta „MA“). MA metodu kiekvienoje iteracijoje iš turimo pirmumo sąrašo, pritaikius elementarias operacijas, gaunamas kitas pirmumo sąrašas, kuris patvirtinamas arba atmetamas. Kadangi vienos iteracijos atlikimui visuose algoritmuose reikia tokių pat kompiuterio laiko resursų, todėl kaip pagrindinis efektyvumo kriterijus buvo paimta tikslo funkcijos geriausia pasiektoji reikšmė. Geriausia pasiekta (pažymėta „ger“) tikslo funkcijos reikšmė fiksuojama po 100, 200, 300, 500 ir 1000 iteracijų. Optimizacija MA metodo buvo pradedama nuo atsitiktinio pirmumo sąrašo. 7.1-7.4 paveikslėliuose pateikti grafikai rodo, jog MA metodo pranašumas ženkliai išryškėja jau po nedidelio skaičiaus iteracijų. MA metodu pasiektas tikslo funkcijos reikšmių vidurkis (100 nusileidimų) mažai skiriasi nuo pasiektos geriausios reikšmės. Tai tik patvirtina metodo patikimumą, nes nuokrypis (išsibarstymas) nuo geriausios reikšmės nėra didelis.



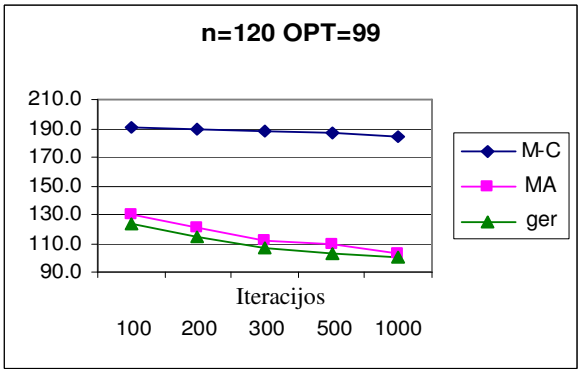
7.1 pav. Tikslo funkcijos reikšmės konvergavimas į global. minimumą (užduočių skaičius *n* = 30, optimumas Opt = 41)



7.3 pav. Tikslo funkcijos reikšmės konvergavimas į global. minimumą (užduočių skaičius *n* = 90, optimumas Opt = 67)



7.2 pav. Tikslo funkcijos reikšmės konvergavimas į global. minimumą (užduočių skaičius *n* = 60, optimumas Opt = 77)

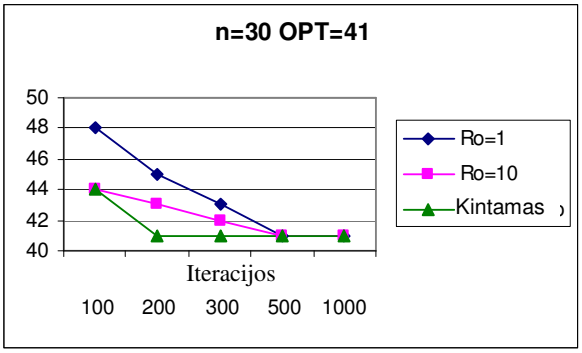


7.4 pav. Tikslo funkcijos reikšmės konvergavimas į global. minimumą (užduočių skaičius *n* = 120, optimumas Opt = 99)

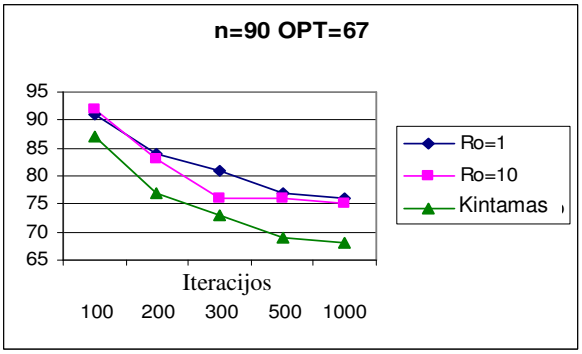
Kadangi modeliuojamojo atkaitinimo algoritmo pranašumas akivaizdžiai išryškėja jau po nedidelio skaičiaus iteracijų, buvo ištirtas ir aplinkos gylio reguliavimas optimizuojant MA metodu.

Elementarių operacijų skaičius kiekvienoje iteracijoje nulemia, kokią galimų sprendinių erdvę mes galime sugeneruoti. Buvo atlikta optimizacija su fiksuotu elementarių operacijų skaičiumi kiekvienoje optimizavimo procedūros iteracijoje (*Ro* = 1 ir *Ro* = 10. Žiūr. 7.5-7.8 pav.). Nedidelis elementarių operacijų, kurios leidžia formuoti turimo sprendinio aplinkas, skaičius formuoja naujus sprendinius „nedideliu atstumu“ nuo einamojo. Todėl optimumo paieška užtrunka ilgiau – reikia didelio iteracijų skaičiaus, kad būtų įmanoma priartėti prie optimalaus sprendinio. Antra vertus, fiksuotas didelis elementarių operacijų skaičius kiekvienoje iteracijoje nors pradžioje ir suteikia galimybę sugeneruoti galimus sprendinius „didesniu atstumu“ nuo einamojo (daugiau šansų „padengti“ sprendinių erdvę), bet vis tik optimizavimo eigoje, kai artėjama prie optimumo, per didelė kaita dažniausiai tik prailgina optimumo paiešką. Juo labiau, jog ir sugeneruotų sąrašų leistinumo tikrinimai užtrunka ilgiau. Aplinkos gylio reguliavimas priklausomai nuo iteracijų skaičiaus leido pagerinti konvergavimo greitį. (Pažymėta „Kintamas Ro“ 7.5-7.8 paveikslėliuose). Šis aplinkos gylio reguliavimas kai kuriais atvejais leido pasiekti maksimalų rezultatą (rasti žinomą optimumą) jau po nedidelio skaičiaus iteracijų.

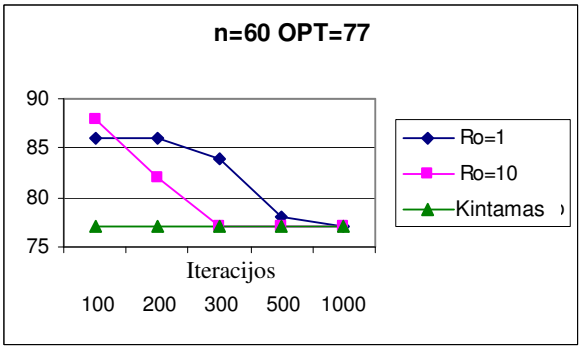
7.5 lentelėje pateikti modeliuojamojo atkaitinimo algoritmo su kintama aplinka testavimo rezultatai, sprendžiant pilną rinkinį testinių uždavinių iš PSPLib. $n = 30$ ir $n = 60$ užduočių atvejais buvo sprendžiama po 480 testinių uždavinių, 120 užduočių atveju – 600 testinių uždavinių. Buvo atlikta po 1000 ir 5000 iteracijų (1000 ir 5000 peržiūrėtų pirmumo sąrašų). Palyginimui yra pateiktas vidutinis sprendimo laikas sekundėmis.



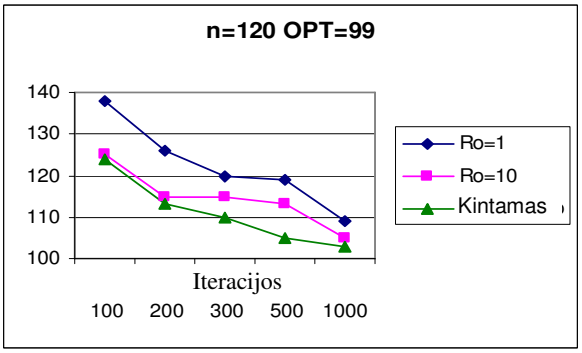
7.5 pav. Tikslo funkcijos reikšmės konvergavimas į globalųjį minimumą, naudojant tris aplinkos tipus (užduočių skaičius n = 30, optimumas Opt = 41)



7.7 pav. Tikslo funkcijos reikšmės konvergavimas į globalųjį minimumą, naudojant tris aplinkos tipus (užduočių skaičius n = 90, optimumas Opt = 67)



7.6 pav. Tikslo funkcijos reikšmės konvergavimas į globalųjį minimumą, naudojant tris aplinkos tipus (užduočių skaičius n = 60, optimumas Opt = 77)



7.8 pav. Tikslo funkcijos reikšmės konvergavimas į globalųjį minimumą, naudojant tris aplinkos tipus (užduočių skaičius n = 120, optimumas Opt = 99)

7.5 lentelė. Modeliuojamojo atkaitinimo algoritmo su kintama aplinka testavimo rezultatai. Testiniuose uždaviniuose su n = 30, 60, 120 užduočių buvo atlikta po 1000 ir 5000 iteracijų (1000 ir 5000 peržiūrėtų pirmumo sąrašų)

	Vid. it. skaičius optimumui ar geriausiai žin. reikšmei rasti	Vid. Laikas, sek.	Kiek kartų surasta geriausia žinoma reikšmė ar optimumas	Geriausios žinomos reikšmės ar optimumo radimas, %	Vidutinis nuokrypis nuo geriausios žinomos reikšmės ar opt., %
n = 30					
IT = 1000	286	0,76	448 iš 480	93,33	0,21
IT = 5000	312	0,84	476 iš 480	99,17	0,08
n = 60					
IT = 1000	886	17,24	128 iš 480	26,67	4,42
IT = 5000	948	19,32	376 iš 480	78,33	2,46
n = 120					
IT = 1000	1000	116,45	1 iš 600	0,17	18,27
IT = 5000	5000	588,37	52 iš 600	10,83	7,62

7.3 RITSU sprendimo genetiniu algoritmu rezultatai

Genetinio algoritmo efektyvumas ir konvergavimo greitis buvo tiriami statistinio modeliavimo būdu. Sukurtasis algoritmas testuojamas naudojant duomenų rinkinius iš tvarkaraščių uždavinių bibliotekos PSPLib. Tam, kad ištirti tikslo funkcijos reikšmės konvergavimo į globalųjį minimumą (ar į geriausią žinomą reikšmę) greitį naudojant genetinį algoritmą, buvo paimta keletas skirtingos dimensijos (skirtingo užduočių skaičiaus) ir sudėtingumo (skirtingo reikiamų resursų skaičiaus bei skirtingo nuoseklumo sąryšių skaičiaus) pradinių duomenų rinkinių. Konvergavimo greičių vidurkiai buvo suskaičiuoti pagal 100 modeliavimo eksperimentų rezultatus, kai naudojami duomenų rinkiniai su 30 ir 60 užduočių, ir pagal 10 modeliavimo eksperimentų rezultatus, kai naudojami duomenų rinkiniai su 90 ir 120 užduočių. 7.6-7.8 lentelėse „it.“ žymi iteracijas.

7.6 lentelė. Tikslo funkcijos reikšmės konvergavimo greitis į globalųjį minimumą naudojant genetinį algoritmą (3 skirtingi pradinių duomenų atvejai iš PSPLib, 4 skirtingi GA parametrų nustatymai, užduočių skaičius n = 30)

	Duomenų rinkinys #1				**Duomenų rinkinys #2**				**Duomenų rinkinys #3**			
Užduočių skaičius	**30**				**30**				**30**			
Žinomas tikslo funkcijos optimumas	**41**				**38**				**63**			
Vidutinis it. skaičius optimumui rasti	83	15	21	18	162	42	35	18	24	6	8	5
Mutavimo tikimybė P_{mut}	0,1	0,1	**0,3**	0,3	0,1	0,1	**0,3**	0,3	0,1	0,1	**0,3**	0,3
Individų populiacijoje skaičius	20	40	**20**	40	20	40	**20**	40	20	40	**20**	40
Vidutinis laikas optimumui rasti, s	3,17	1,38	**0,99**	1,33	6,04	3,28	**1,48**	1,58	1,51	0,65	**0,48**	0,61

7.7 lentelė. Tikslo funkcijos reikšmės konvergavimo greitis į geriausią žinomą reikšmę naudojant genetinį algoritmą (3 skirtingi pradinių duomenų atvejai iš PSPLib, 4 skirtingi GA parametrų nustatymai, užduočių skaičius n = 60)

	Duomenų rinkinys #1				**Duomenų rinkinys #2**				**Duomenų rinkinys #3**			
Užduočių skaičius	**60**				**60**				**60**			
Geriausia žinoma tikslo funkcijos reikšmė	**77**				**59**				**76**			
Vidut. it. skaičius geriausiam sprendiniui rasti	110	92	45	32	358	188	345	175	892	141	38	57
Mutavimo tikimybė P_{mut}	0,1	0,1	**0,3**	0,3	0,1	0,1	**0,3**	0,3	0,1	0,1	**0,3**	0,3
Individų populiacijoje skaičius	20	40	**20**	40	20	40	**20**	40	20	40	**20**	40
Vidutinis laikas geriausiam sprendiniui rasti, s	23,8	39,9	**10,7**	14,4	77,8	80,7	**75,3**	76,1	193,4	60,6	**8,3**	24,8

7.8 lentelė. Tikslo funkcijos reikšmės konvergavimo greitis į geriausią žinomą reikšmę naudojant genetinį algoritmą (1 pradinių duomenų atvejis, kai n = 90, 2 atvejai, kai n = 120, 4 skirtingi GA parametrų nustatymai)

	Duomenų rinkinys #1				**Duomenų rinkinys #1**				**Duomenų rinkinys #2**			
Užduočių skaičius	**90**				**120**				**120**			
Geriausia žinoma tikslo funkcijos reikšmė	**67**				**99**				**111**			
Vidutinis it. skaičius geriausiam sprendiniui rasti	*	*	944	452	*	*	894	442	*	*	*	436
Mutavimo tikimybė P_{mut}	0,1	0,1	0,3	**0,3**	0,1	0,1	0,3	**0,3**	0,1	0,1	0,3	**0,3**
Individų populiacijoje skaičius	20	40	20	**40**	20	40	20	**40**	20	40	20	**40**
Vidutinis laikas geriausiam sprendiniui rasti, s	*	*	636	**623**	*	*	1388	**1368**	*	*	*	**1350**

* – geriausia žinoma tikslo funkcijos reikšmė nebuvo pasiekta (modeliuojant buvo atlikta maksimum 1000 iteracijų).

Modeliavimo rezultatai parodė, jog tais atvejais, kai uždavinių dimensija mažesnė (užduočių skaičius yra 30 arba 60), geresnius rezultatus galime gauti naudodami mutacijos tikimybę

$P_{mut} = 0.3$ ir mažesnį chromosomų (individų) populiacijoje skaičių (20). Per maža mutacijos tikimybė negarantuoja pakankamos pirmumo sąrašų rekombinacijos, o didesnis chromosomų (individų) skaičius populiacijoje reikalauja daugiau skaičiavimo laiko (7.6 ir 7.7 lentelės). Tais atvejais, kai uždavinių dimensija didesnė (užduočių skaičius 90 arba 120), naudojant per mažą mutacijos tikimybę ir mažesnį chromosomų (individų) populiacijoje skaičių, optimumo suradimui gali tekti eikvoti daug skaičiuojamųjų laiko resursų (7.8 lentelė). Nors su didesniu chromosomų skaičiumi (40) vienos iteracijos skaičiavimai trunka ilgiau, tačiau geresnė sprendinių rekombinacija viso proceso eigoje duoda geresnį galutinį skaičiavimų rezultatą – geriausios žinomos tikslo funkcijų reikšmės surandamos greičiau.

Išspręsta visa uždavinių klasė su 30 ir 60 užduočių. 30 užduočių atveju ištirta mutavimo tikimybės įtaka genetinio algoritmo efektyvumui, kai mutacijai atrinktoje chromosomoje (pirmumo sąraše) dvi atsitiktinės užduotys sukeičiamos vietomis arba viena užduotis perkeliama į kitą poziciją. Atliekant mutacijos operaciją yra tikrinama, jog nauji pirmumo sąrašai būtų leistini.

Buvo atlikti eksperimentai ir su kitokia mutacijos interpretacija. Jei mutavimo procedūrai chromosomos buvo atrenkamos su duota tikimybe, tai tokiu atveju, ne su visomis chromosomomis mutacija buvo atliekama. Eksperimentai su 30 užduočių testiniais uždaviniais parodė (7.9 lentelė), jog kai mutacijos tikimybė $P_{mut} > 0,3$, visi testinių uždavinių žinomi optimumai yra surandami, bet šios tikimybės didinimas (t.y. didinant chromosomų, atrinktų kryžminimui, skaičių) leidžia žinomus optimumus surasti per mažesnį iteracijų skaičių ir per trumpesnį laiką.

7.9 lentelė. Genetinio algoritmo mutacijos tikimybės įtaka algoritmo efektyvumui. Užduočių skaičius *n* = 30, chromosomų (individų) skaičius populiacijoje KiekChrom = 20, maksimalus leistinas iteracijų skaičius Max_IT = 1000

	Iš 480 uždavinių atvejų		
P_{mut}	Rasti optimumai	Vid. iteracijų skaičius optimumui rasti	Vid. laikas optimumui rasti, (s)
0,3	99,58%	41	1,19
0,4	100%	30	0,77
0,5	100%	22	0,48
0,6	100%	22	0,45
0,7	100%	18	0,29
0,8	100%	17	0,26
0,9	100%	16	0,24
1,0	**100%**	**16**	**0,21**

Buvo atlikti tyrimai, kai P_{mut} yra geno mutavimo tikimybė (genai chromosomoje – tai užduočių numeriai pirmumo sąraše). Vienoje iteracijoje generuojama 20 pirmumo sąrašų (20 tvarkaraščių). Todėl per 50 algoritmo iteracijų yra patikrinama iki 1000 pirmumo sąrašų (atitinkamai, per 250 algoritmo iteracijų patikrinama iki 5000 pirmumo sąrašų). Keičiant mutavimo tikimybę P_{mut}, buvo tiriama, kiek vidutiniškai iteracijų (vienoje iteracijoje peržiūrima keletas pirmumo sąrašų, todėl pateiktas ir vidutinis peržiūrėtų pirmumo sąrašų skaičius) reikia optimumui ar geriausiai žinomai

tikslo funkcijos reikšmei surasti, kiek vidutiniškai tam reikia laiko, kiek kartų vidutiniškai randamas optimumas ar geriausia žinoma tikslo funkcijos reikšmė (taip pat suskaičiuota kiek procentų atvejų tai pavyko padaryti). 30 užduočių atveju suskaičiuotas surastos tikslo funkcijos reikšmės vidutinis nuokrypis nuo žinomo optimumo, o 60 ir 120 užduočių atveju – vidutinis nuokrypis nuo geriausios žinomos tikslo funkcijos reikšmės. Šio tyrimo rezultatai pateikti 7.10-7.15 lentelėse. Iš šio tyrimo matyti, jog 30 užduočių atveju geriausi rezultatai pasiekiami su genų mutavimo tikimybe $P_{mut} = 0{,}08$. Visi rodikliai (vidutinis reikiamas iteracijų skaičius, sugaištas laikas, optimumo suradimų skaičius ar procentas bei vidutinis nuokrypis nuo optimumo) yra geresni ar neblogesni už rodiklius su kitokia mutavimo tikimybe (žiūr. 7.10-7.11 lenteles). 60 užduočių atveju, kai peržiūrima iki 1000 pirmumo sąrašų (žiūr. 7.12 lentelę), taip pat galima pastebėti $P_{mut} = 0{,}08$ pranašumą pagal visus rodiklius. Bet didinant iteracijų (peržiūrėtų pirmumo sąrašų) skaičių, išryškėja $P_{mut} = 0{,}05$ pranašumas (žiūr. 7.13 lentelę). 120 užduočių atveju, geriausi rezultatai pasiekiami su genų mutavimo tikimybe $P_{mut} = 0{,}03$ (žiūr. 7.14-7.15 lenteles).

7.10 lentelė. Užduočių skaičius *n* = 30, chromosomų skaičius populiacijoje KiekChrom = 20, maksimalus leistinas iteracijų skaičius Max_IT = 50 (maximum 1000 peržiūrėtų pirmumo sąrašų)

P_{mut}	Vid. iteracijų skaičius optimumui rasti	Vid. Laikas, sek.	Surastas optimumas, kiek kartų iš 480	Optimumo radimas, %	Vidutinis nuokrypis nuo opt., %
0,01	22 (440)	0,92	391	81,46	0,833
0,03	16 (320)	0,74	449	93,54	0,205
0,05	14 (280)	0,72	467	97,29	0,104
0,08	**13 (260)**	**0,66**	**474**	**98,75**	**0,036**
0,09	**13 (260)**	**0,69**	**472**	**98,33**	**0,040**
0,10	**13 (260)**	**0,70**	**471**	**98,13**	**0,043**
0,11	14 (280)	0,74	469	97,71	0,064
0,12	14 (280)	0,76	467	97,29	0,068
0,15	14 (280)	0,82	464	96,67	0,096
0,20	15 (300)	0,95	452	94,17	0,162
0,30	18 (360)	1,36	418	87,08	0,372

7.11 lentelė. Užduočių skaičius *n* = 30, chromosomų skaičius populiacijoje KiekChrom = 20, maksimalus leistinas iteracijų skaičius Max_IT = 250 (maximum 5000 peržiūrėtų pirmumo sąrašų)

P_{mut}	Vid. iteracijų skaičius optimumui rasti	Vid. Laikas, sek.	Surastas optimumas, kiek kartų iš 480	Optimumo radimas, %	Vidutinis nuokrypis nuo opt., %
0,03	18 (360)	0,90	479	99,79	0,008
0,05	15 (300)	0,78	480	100,00	0,000
0,08	**14 (280)**	**0,71**	**480**	**100,00**	**0,000**
0,09	**14 (280)**	**0,73**	**480**	**100,00**	**0,000**
0,10	**14 (280)**	**0,74**	**480**	**100,00**	**0,000**
0,11	15 (300)	0,78	480	100,00	0,000
0,12	15 (300)	0,80	479	99,79	0,004
0,15	16 (320)	0,89	478	99,58	0,008
0,20	21 (420)	1,29	476	99,17	0,023
0,30	30 (600)	2,17	464	96,67	0,088

7.12 lentelė. Užduočių skaičius *n* = 60, chromosomų skaičius populiacijoje KiekChrom = 20, maksimalus leistinas iteracijų skaičius Max_IT = 50 (maximum 1000 peržiūrėtų pirmumo sąrašų)

P_{mut}	Vid. iteracijų skaičius ieškant geriausios žinomos reikšmės	Vid. Laikas, sek.	Kiek kartų iš 480 surasta geriausia žinoma reikšmė	Geriausios žinomos reikšmės suradimas, %	Vidutinis nuokrypis nuo geriausios žinomos reikšmės., %
0,03	42 (840)	14,07	201	41,88	2,5128
0,05	41 (820)	13,51	208	43,33	2,4813
0,07	**41 (820)**	**12,74**	**221**	**46,04**	**2,4554**
0,08	**41 (820)**	**11,94**	**231**	**48,13**	**2,4066**
0,09	**41 (820)**	**12,52**	**212**	**44,17**	**2,5904**
0,10	42 (840)	12,98	193	40,21	2,8327
0,11	43 (860)	14,96	183	38,12	2,9235
0,12	43 (860)	15,13	182	37,92	2,9867
0,15	44 (880)	16,77	131	27,29	3,8425

7.13 lentelė. Užduočių skaičius *n* = 60, chromosomų skaičius populiacijoje KiekChrom = 20, maksimalus leistinas iteracijų skaičius Max_IT = 250 (maximum 5000 peržiūrėtų pirmumo sąrašų)

P_{mut}	Vid. iteracijų skaičius ieškant geriausios žinomos reikšmės	Vid. Laikas, sek.	Kiek kartų iš 480 surasta geriausia žinoma reikšmė	Geriausios žinomos reikšmės suradimas, %	Vidutinis nuokrypis nuo geriausios žinomos reikšmės., %
0,03	**83 (1660)**	**26,01**	**428**	**89,17**	**0,2731**
0,05	**79 (1580)**	**25,50**	**446**	**92,92**	**0,1637**
0,07	**84 (1680)**	**26,08**	**433**	**90,21**	**0,2269**
0,08	**89 (1780)**	**24,88**	**424**	**88,33**	**0,2613**
0,09	96 (1920)	26,16	409	85,21	0,3389
0,10	100 (2000)	33,54	395	82,29	0,4480
0,11	105 (2100)	33,89	381	79,37	0,5485
0,12	113 (2260)	34,77	373	77,71	0,6002
0,15	116 (2320)	36,22	351	73,12	0,7824

7.14 lentelė. Užduočių skaičius *n* = 120, chromosomų skaičius populiacijoje KiekChrom = 20, maksimalus leistinas iteracijų skaičius Max_IT = 50 (maximum 1000 peržiūrėtų pirmumo sąrašų)

P_{mut}	Vid. iteracijų skaičius ieškant geriausios žinomos reikšmės	Vid. Laikas, sek.	Kiek kartų iš 600 surasta geriausia žinoma reikšmė	Geriausios žinomos reikšmės suradimas, %	Vidutinis nuokrypis nuo geriausios žinomos reikšmės., %
0,01	50 (1000)	108,99	0	0	24,43
0,02	50 (1000)	108,37	1	0,17	19,62
0,03	**50 (1000)**	**107,85**	**1**	**0,17**	**16,35**
0,04	**50 (1000)**	**111,10**	**1**	**0,17**	**16,64**
0,05	**50 (1000)**	**117,66**	**0**	**0**	**16,70**
0,06	50 (1000)	118,35	0	0	16,81
0,07	50 (1000)	119,99	1	0,17	17,21
0,08	50 (1000)	123,26	1	0,17	17,45
0,09	50 (1000)	127,38	0	0	17,81
0,10	50 (1000)	130,21	0	0	18,67

7.15 lentelė. Užduočių skaičius *n* = 120, chromosomų skaičius populiacijoje KiekChrom = 20, maksimalus leistinas iteracijų skaičius Max_IT = 250 (maximum 5000 peržiūrėtų pirmumo sąrašų)

p_{mut}	Vid. iteracijų skaičius ieškant geriausios žinomos reikšmės	Vid. Laikas, sek.	Kiek kartų iš 600 surasta geriausia žinoma reikšmė	Geriausios žinomos reikšmės suradimas, %	Vidutinis nuokrypis nuo geriausios žinomos reikšmės., %
0,01	242 (4840)	514,05	58	9,67	7,44
0,02	**229 (4580)**	**463,55**	**141**	**23,50**	**4,65**
0,03	**225 (4500)**	**469,25**	**155**	**25,83**	**4,02**
0,04	**226 (4520)**	**483,80**	**144**	**24,00**	**4,15**
0,05	**231 (4620)**	**535,45**	**118**	**19,67**	**4,89**
0,06	234 (4680)	542,32	97	16,17	5,44
0,07	238 (4760)	554,70	74	12,33	6,33
0,08	242 (4840)	576,14	43	7,17	7,20
0,09	244 (4880)	593,91	32	5,33	7,91
0,10	246 (4920)	616,48	28	4,67	8,92

7.16 lentelėje pateikti vidutiniai tikslo funkcijos reikšmių skirtumai tarp kritinių kelių metodu (naudojant priešsrovio algoritmą) gautos reikšmės bei metaeuristiniais metodais gautų tikslo funkcijos reikšmių, peržiūrėjus 100 ir 1000 pirmumo sąrašų.

Vidutinis tikslo funkcijos reikšmės pagerėjimas = $\frac{1}{U}\sum_{i=1}^{U}\frac{T^i_{krit.k.} - T^i_{eurist.}}{T^i_{krit.k.}}$,

čia $T^i_{krit.k.}$ – kritinių kelių metodu gauta tikslo funkcijos reikšmė, sprendžiant *i*-tąjį uždavinį, $T^i_{eurist.}$ – metaeuristiniu metodu gauta tikslo funkcijos reikšmė, sprendžiant *i*-tąjį uždavinį, U – skirtingų uždavinių skaičius (U = 480, kai n = 30 ir n = 60, U = 600, kai n = 120).

7.16 lentelė. Tikslo funkcijos reikšmės pagerėjimas nuo kritinių kelių metodu gautos reikšmės po tam tikro skaičiaus peržiūrėtų pirmumo sąrašų

Užduočių skaičius	MA algoritmu gautas vidutinis pagerėjimas, %		GA algoritmu gautas vidutinis pagerėjimas, %	
	Po 100 peržiūrėtų pirm. sąrašų	Po 1000 peržiūrėtų pirm. sąrašų	Po 100 peržiūrėtų pirm. sąrašų	Po 1000 peržiūrėtų pirm. sąrašų
n = 30	14,85	29,42	19,98	30,21
n = 60	13,36	24,95	19,28	26,36
n = 120	10,69	23,84	15,82	25,38

7.4 Metaeuristinių algoritmų efektyvumo palyginimas

Sukurtų algoritmų efektyvumas buvo lyginamas su jau žinomų algoritmų taikymo toms pačioms problemoms spręsti rezultatais (Hartmann, 2002). 7.17 lentelėje palyginti įvairių algoritmų rezultatai, kai žinomas tikslo funkcijos optimumas.

7.17 lentelė. Vidutinis nuokrypis (%) nuo žinomo optimalaus projekto užbaigimo laiko, kai užduočių skaičius n = 30

Algoritmas	Autoriai, metai	Iteracijų skaičius	
		1000	5000
Sukurtasis GA		**0,04**	**0.00**
MA su kint.aplinka		**0,21**	**0,08**
Self-adapting GA	(Hartmann, 2002)	0,38	0,22
simulated annealing	(Bouleimen, Lecocq, 1998)	0,38	0,23
activity list GA	(Hartmann, 1998)	0,54	0,25
adaptive sampling	(Schirmer, 2000)	0,65	0,44
tabu search	(Baar, Brucker, Knust, 1999)	0,86	0,44
adaptive sampling	(Kolisch, Drexl, 1996)	0,74	0,52
serial sampling (LFT)	(Kolisch, 1996b)	0,83	0,53
random key GA	(Hartmann, 1998)	1,03	0,56
priority rule GA	(Hartmann, 1998)	1,38	1,12
parallel sampling (LFT)	(Kolisch, 1996b)	1,40	1,29
problem space GA	(Leon, Ramamoorthy, 1995)	2,08	1,59

Testinių uždavinių su 60 ir 120 užduočių atvejais nėra žinomi tikslo funkcijos optimumai. Rezultatus galima palyginti su kritinio kelio apatine riba, taip pat su geriausiomis žinomomis tikslo funkcijos reikšmėmis. 7.18 ir 7.19 lentelėse pateikti lyginamieji tikslo funkcijos nuokrypiai nuo

kritinio laiko, o 7.20 lentelėje pateikti lyginamieji nuokrypiai nuo geriausios žinomos tikslo funkcijos reikšmės.

7.18 lentelė. Vidutinis tikslo funkcijos reikšmės nuokrypis (%) nuo kritinio laiko, kai užduočių skaičius $n = 60$

Algoritmas	Autoriai, metai	Iteracijų skaičius	
		1000	5000
Sukurtasis GA		**10,14**	**9,38**
MA su kint.aplinka		12,41	**11,23**
Self-adapting GA	(Hartmann, 2002)	12,21	11,70
activity list GA	(Hartmann, 1998)	12,68	11,89
simulated annealing	(Bouleimen, Lecocq, 1998)	12,75	11,90
adaptive sampling	(Schirmer, 2000)	12,94	12,59
priority rule GA	(Hartmann, 1998)	13,30	12,74
adaptive sampling	(Kolisch, Drexl, 1996)	13,51	13,06
parallel sampling (LFT)	(Kolisch, 1996b)	13,59	13,23
random key GA	(Hartmann, 1998)	14,68	13,32
tabu search	(Baar, Brucker, Knust, 1999)	13,80	13,48
problem space GA	(Leon, Ramamoorthy, 1995)	14,33	13,49
serial sampling (LFT)	(Kolisch, 1996b)	13,96	13,53

7.19 lentelė. Vidutinis tikslo funkcijos reikšmės nuokrypis (%) nuo kritinio laiko, kai užduočių skaičius $n = 120$

Algoritmas	Autoriai, metai	Iteracijų skaičius	
		1000	5000
Sukurtasis GA		**35,61**	**33,21**
MA su kint.aplinka		**38,73**	**35,57**
Self-adapting GA	(Hartmann, 2002)	37,19	35,39
activity list GA	(Hartmann, 1998)	39,37	36,74
simulated annealing	(Bouleimen, Lecocq, 1998)	42,81	37,68
priority rule GA	(Hartmann, 1998)	39,93	38,49
adaptive sampling	(Schirmer, 2000)	39,85	38,70
parallel sampling (LFT)	(Kolisch, 1996b)	39,60	38,75
adaptive sampling	(Kolisch, Drexl, 1996)	41,37	40,45
problem space GA	(Leon, Ramamoorthy, 1995)	42,91	40,69
serial sampling (LFT)	(Kolisch, 1996b)	42,84	41,84
random key GA	(Hartmann, 1998)	45,82	42,25

7.20 lentelė. Vidutinis nuokrypis (%) nuo geriausio žinomo projekto užbaigimo laiko

Algoritmas	Užd. skaičius	Iteracijų skaičius	
		1000	5000
MA su kint. aplinka	$n = 60$	4,42	2,46
	$n = 120$	18,27	7,62
GA besiremiantis pirmumo sąrašu	$n = 60$	**2,41**	**0,16**
	$n = 120$	16,35	**4,02**
Self-adapting GA (Hartmann, 2002)	$n = 60$	3,26	2,88
	$n = 120$	**9,69**	8,33

Sukurtasis GA efektyvumu pranoksta žinomus GA algoritmus (pvz. Self-adapting GA). Testavimo rezultatai parodė, jog sukurtasis GA veikia efektyviau tiek prie didelio, tiek prie mažo iteracijų skaičiaus, kai $n \leq 60$. Kai $n = 120$, algoritmo efektyvumas išryškėja didinant iteracijų skaičių, t.y. kai peržiūrima daugiau pirmumo sąrašų.

Sukurtasis modeliuojamojo atkaitinimo algoritmas su kintama aplinka, kai $n \leq 60$, efektyvumu lenkia ne tik Bouleimen K. ir Lecocq H. pasiūlytą MA, bet ir „Self-adapting GA“.

Šiame darbe, o taip pat kitų autorių gauti rezultatai rodo, jog populiaciniai algoritmai (pvz. genetinis) leidžia gauti geresnius sprendinius, palyginus su nuosekliaisiais algoritmais (pvz. modeliuojamojo atkaitinimo).

7.5 Nuoseklios ir evoliucinės paieškos algoritmų realizavimo aspektai

Sudaryta programinė įranga programinių sistemų Delphi, FreePascal, C++, MathCad priemonėmis, RITSU uždaviniams spręsti metaeuristiniais algoritmais, pritaikant darbų pirmumo sąrašą bei sprendinio aplinkos reguliavimą. Jos aprašymas pateiktas priede E. Ją sudaro apie 20 funkcijų ir procedūrų. Sukurtosios funkcijos bei procedūros yra sudarytos taip, kad jas būtų galima nesunkiai pritaikyti realizuojant įvairias metaeuristinės paieškos paradigmas. Aptarsime RITSU sprendimo metaeuristinių metodų tobulinimo ir praktinio realizavimo aspektus.

Užduočių pirmumo sąrašo panaudojimas nuoseklios paieškos algoritmuose leidžia efektyviai pereiti nuo vieno sprendinio prie kito. Paieškos su draudimais atveju, pirmumo sąrašas gali būti panaudotas draudimų sąrašui konstruoti bei modifikuoti. Paieška gali tapti efektyvesnė, jei į draudimų sąrašą būtų įtraukiamas neefektyvus sprendinys kartu su savo aplinka, įvedant topogiją, kuri panaudota 6-ajame skyriuje modeliuojamojo atkaitinimo algoritmui konstruoti. Tokiu atveju į draudimų sąrašą būtų įtraukiami visi sąrašai iš tam tikro spindulio rutulio, t.y. gaunami iš uždraustojo pirmumo sąrašo atlikus tam tikrą elementarių operacijų skaičių (žiūr. 3.9.2, 6.6 skyrelius ir B priedą). Optimizavimo metu aplinkos spindulys gali būti reguliuojamas priklausomai nuo iteracijos numerio bei tikslo funkcijos pagerėjimo.

Evoliucinės paieškos algoritmuose, pavyzdžiui, išbarstytosios paieškos, dalelių spiečių algoritmuose ir kituose, pirmumo sąrašo ir kintamos aplinkos metodus taip pat galima efektyviai taikyti sprendinių populiacijoms generuoti ir modifikuoti. Metodiškos deterministinės paieškos metu galima konstruoti atraminį rinkinį pirmumo sąrašų, kurie gali būti gaunami vienas iš kito atlikus ne mažiau, negu nustatyta elementarių operacijų. Šis nustatytas elementarių operacijų skaičius optimizavimo metu yra reguliuojamas priklausomai nuo iteracijos numerio ir nuo sprendinių, sukonstruotų remiantis atraminiais, efektyvumo.

Aptarsime kombinatorinių uždavinių sprendimo stochastinių metaeuristinių algoritmų stabdymo problemą. Kadangi daugelyje tokių uždavinių nėra žinomos paprastos sprendinio optimalumo sąlygos, tai dažniausiai algoritmai yra stabdomi, atlikus iš anksto nustatytą iteracijų skaičių. Tačiau šis kriterijus nėra patogus, kai optimumas gali būti jau rastas atlikus mažesnį nei nustatytas iteracijų skaičių. Priimant sprendimą apie optimumo radimą, galima pasinaudoti informacija apie geriausias pasiektas tikslo funkcijos reikšmes, gautas optimizavimo metu. Tegul $\eta_{(1)},\eta_{(2)},...,\eta_{(k)}$ yra k geriausių tikslo funkcijos reikšmių, apskaičiuotų per tam tikrą skaičių nuoseklios paieškos

algoritmo iteracijų. Jei algoritmas yra stochastinis, tai $\eta_{(1)},\eta_{(2)},...,\eta_{(k)}$ galima laikyti tikslo funkcijos reikšmių sekos, apskaičiuotos optimizavimo metu, pozicinėmis statistikomis, ir apskaičiuoti tikslo funkcijos minimalios reikšmės tiesinį įvertį bei pasikliautinąjį intervalą (Žilinskas, Zhigliavsky, 1991, Sakalauskas, Bartkutė , Felinskas, 2006). Šis būdas buvo pritaikytas pakavimo uždaviniui spręsti modeliuojamojo atkaitinimo metodu. Kompiuterinio modeliavimo būdu buvo parodyta, kad minimumo tiesinio įverčio koeficientus bei pasikliautinąjį intervalą galima apskaičiuoti naudojantis nepriklausomų atsitiktinių dydžių pozicinių statistikų teorija (Sakalauskas, Bartkutė, Felinskas, 2006). Nuoseklios paieškos algoritmų stabdymo taisyklių sudarymas remiantis informacija apie geriausias pasiektas tikslo funkcijos reikšmes dar reikalauja papildomų tyrimų.

7.6 HP montažinių plokščių surinkimo uždavinys

Sukurtieji algoritmai pritaikyti praktiniams taikomiesiems tvarkaraščių uždaviniams spręsti.

Kompanijoje „Hewlett – Packard (HP)“ susiduriama su montažinių plokščių (Printed Circuit Board, PCB) surinkimo uždaviniu. Šiame uždavinyje reikia paskirstyti operacijas, surinkinėjant montažines plokštes (MP). Sprendžiant šį taikomąjį uždavinį susiduriama su kombinatoriniu uždaviniu, kuris literatūroje apibrėžiamas kaip $P|\#\leq k|C_{max}$ (žiūr. 4.1.5 skyrelį). Tai užduočių paskirstymo keliems lygiagretiems procesoriams uždavinys.

Uždavinio $P|\#\leq k|C_{max}$ sprendimas gali būti pritaikytas robotizuotose surinkimo linijose. Hillier ir Brandeau tyrinėjo operacijų skirstymo uždavinį, taikant spausdintuvo montažinių plokščių surinkimo procese kompanijoje „Hewlett – Packard (HP)“ (Hillier, Brandeau, 1998). Pateikti eksperimentų rezultatai naudojant duomenų rinkinius, paremtus pavyzdžiais, kuriuos pateikė kompanija HP. Kiekvieną šių eksperimentų duomenų rinkinį sudarė:

b – skirtingų tipų plokščių skaičius;

c – komponentų tipų skaičius;

d_i – poreikis i-ojo tipo plokščių $(i = 1, ... , b)$;

v_{ij} – skaičius j-ojo tipo komponentų, kurie turi būtį įmontuoti i-ojo tipo plokštėje;

p – apdorojimo laikas, kurio reikia komponentui (bet kurio tipo) įmontuoti;

k – maksimalus skaičius komponentų tipų, kurie gali būti paskirti bet kuriam procesoriui.

Pagal tuos montažinių plokščių surinkimo duomenų rinkinius buvo paruošti dviejų tipų duomenų rinkiniai (Dell’Amico, Iori, Martello, 2004) uždaviniui $P|\#\leq k|C_{max}$. Jie paruošti taip:

PCB_1: kiekvienam plokštės tipui i ($i = 1, ..., b$) ir komponentų tipui j ($j = 1, ..., c$) nusakytos užduotys, kurių apdorojimo laikas yra $pv_{ij}d_i$;

PCB_2: kiekvienam komponentų tipui j ($j = 1, ..., c$) nusakytos užduotys, kurių apdorojimo laikas yra $p\sum_{i=1}^{b} v_{ij}d_i$.

7.21 lentelė. Modeliavimo rezultatai pagal PCB_1

PCB_1	*n*	*m*	*k*	Geriausia žinoma reikšmė	Pasiekta reikšmė
	403	5	81	115340	115347
	403	10	41	57670	57678
	403	20	21	32274	32276
	1341	5	269	512403	512411
	1341	10	135	256202	256209
	1341	20	68	128101	128110
	1417	5	284	467925	467929
	1417	10	142	233963	233971
	1417	20	71	116982	116985
	1312	5	263	446266	446272
	1312	10	132	223133	223137
	1312	20	66	111567	111271

7.22 lentelė. Modeliavimo rezultatai pagal PCB_2

PCB_2	*n*	*m*	*k*	Geriausia žinoma reikšmė	Pasiekta reikšmė
	314	5	63	114324	114327
	314	10	32	57162	57166
	314	20	16	31965	31969
	972	5	195	512007	512011
	972	10	98	256004	256009
	972	20	49	128002	128012
	1084	5	217	459495	459499
	1084	10	109	229749	229754
	1084	20	55	114876	114882
	1056	5	212	445266	445269
	1056	10	106	222633	222639
	1056	20	53	111317	111325

Aštuoniems atvejams su skirtingu užduočių skaičiumi *n* buvo sudaryti atskiri atvejai su skirtingu procesorių skaičiumi *m* (kai *m = 5, 10, 20*). Kiekvienam iš 24 atvejų su šių dviejų parametrų kombinacija įvestas ribojimas *k = [n/m]* (žiūr. 7.21-7.22 lenteles). Šios 24 reikšmės yra saugomos faile *Kvalues.txt.*

Yra paruošti 8 duomenų failai – *ES1a.txt, ES2a.txt, ES3a.txt, ES4a.txt, ES1b.txt, ES2b.txt, ES3b.txt, ES4b.txt,* užduočių skaičius *n* atitinkamai lygus *403, 1341, 1417, 1312, 314, 972, 1084, 1056*. Kiekvieno iš šių failų struktūra yra analogiška ir juose pateikiama tokia informacija (žiūr. *ES1a.txt*):

403 – užduočių skaičius (*n*);
220 – fiktyvus;
0 – fiktyvus;
570 – 1-osios užduoties apdorojimo laikas;
285 – 2-osios užduoties apdorojimo laikas;
...
13836 – 402-osios užduoties apdorojimo laikas;
3459 – 403-osios (paskutiniosios) užduoties apdorojimo laikas.

Šiam uždaviniui spręsti buvo pritaikytas genetinis algoritmas. Šio algoritmo taikymo rezultatai parodė jo efektyvumą, sprendžiant panašias problemas (žiūr. 7.21-7.22 lentelių paskutinius du stulpelius). Buvo patikrinta po 10000 pirmumo sąrašų ir fiksuojama geriausia pasiekta tikslo funkcijos reikšmė.

7.7 Išvados

1. Modeliuojamojo atkaitinimo, besiremiančio užduočių pirmumo sąrašu, testavimo rezultatai parodė, jog algoritmo efektyvumą galima padidinti atitinkamai reguliuojant aplinkos gylį ir Pareto skirstinio parametrą.
2. Eksperimentai parodė, jog modeliuojamojo atkaitinimo algoritmas su kintama aplinka yra efektyvesnis, už algoritmą su fiksuotu mažu arba dideliu aplinkos „gyliu“.
3. Sprendžiant uždavinius iš testinių pavyzdžių duomenų bazės, testavimo rezultatai parodė, jog algoritmas sėkmingai suranda optimalias ir geriausias žinomas tikslo funkcijos reikšmes per praktiškai priimtiną laiką.
4. Tyrimai parodė, jog geriausių rezultatų galima tikėtis, kai mutacijos tikimybė sukurtajame genetiniame algoritme $P_{mut} = 0{,}08$, jei užduočių skaičius $n = 30$ arba $n = 60$. Šią išvadą patvirtina tyrimų su įvairiomis uždavinių klasėmis rezultatai. Kai užduočių skaičius $60 < n \leq 120$, mutacijos tikimybę reikia mažinti. Geriausi rezultatai pasiekiami, kai $P_{mut} = 0{,}03$.
5. Sukurtųjų algoritmų testavimo rezultatų palyginimas su kitų žinomų metaeuristikos algoritmų taikymo tiems patiems uždaviniams spręsti rezultatais parodė, jog sukurtieji algoritmai leidžia efektyviai spręsti RITSU uždavinius ir kartais leidžia gauti geresnių rezultatų.
6. Sukurtojo genetinio algoritmo realizacijos taikymas HP montažinių plokščių surinkimo uždaviniui spręsti, parodė, jog šis algoritmas gali būti taikomas praktiniams tvarkaraščių uždaviniams spręsti.

8 skyrius. Išvados

Sprendžiant darbe iškeltus uždavinius, yra pasiekti tokie rezultatai:

1. Atliktas RITSU sprendimo klasikiniais ir metaeuristiniais metodais analitinis tyrimas.
2. Pasiūlyta metaeuristinės paieškos metodologija, besiremianti tvarkaraščio kodavimu užduočių pirmumo sąrašu ir nuoseklaus dekodavimo procedūra. Toks tvarkaraščio kodavimas leidžia taikyti įvairius euristinius optimizavimo metodus, suteikia galimybę lengviau generuoti naujus sprendinius iš leistinos sprendinių erdvės, pritaikant elementarias operacijas.
3. Sudarytas atsitiktinės paieškos (Monte-Karlo metodas) tvarkaraščių optimizavimo metodas.
4. Sudarytas modeliuojamojo atkaitinimo algoritmas RITSU uždaviniams spręsti, kuriame realizuotas aplinkos gylio reguliavimas pagal Pareto dėsnį, suderintas su temperatūros reguliavimu bei atsitiktinių sprendinių generavimu.
5. Sukurtas genetinis algoritmas RITSU optimizuoti, realizuojant kryžminimo, mutacijos, atrankos procedūras, pritaikant užduočių pirmumo sąrašus.
6. Programinių sistemų Delphi, FreePascal, C++, MathCad priemonėmis sudaryta programinė įranga RITSU uždaviniams metaeuristiniais algoritmais spręsti, pritaikant užduočių pirmumo sąrašą bei sprendinio aplinkos reguliavimą.
7. Sukurtieji algoritmai yra ištirti statistinio modeliavimo būdu ir palyginti su kitais algoritmais, naudojant duomenų rinkinius ir žinomus sprendinius iš bibliotekos PSPLib.

Remiantis gautais rezultatais bei atliktais tyrimais, galima daryti tokias išvadas:

1. Skaičiuojamojo eksperimento metu nustatyta, kad optimalius tvarkaraščius su turima kompiuterine įranga galima rasti per priimtiną skaičiavimo laiką, kai uždavinio dimensija $n = 30 \div 40$.
2. Sprendžiant uždavinius iš testinių pavyzdžių duomenų bazės, testavimo rezultatai parodė, jog algoritmas sėkmingai suranda optimalias ir geriausias žinomas tikslo funkcijos reikšmes per praktiškai priimtiną laiką (kai uždavinio dimensija $n = 30$, sprendimo laikas ~ 1 sek., $n = 60$, sprendimo laikas ~ 20 sek., $n = 120$, sprendimo laikas ~ 10 min.).
3. Modeliuojamojo atkaitinimo, besiremiančio užduočių pirmumo sąrašu, testavimo rezultatai parodė, jog algoritmo efektyvumas padidėja 15–20%, reguliuojant aplinkos gylio ir Pareto skirstinio parametrus.
4. Genetinio algoritmo su užduočių pirmumo sąrašu tyrimai parodė, jog geriausių rezultatų galima tikėtis, kai mutacijos tikimybė sukurtajame genetiniame algoritme $P_{mut} = 0{,}08$, jei

užduočių skaičius $n = 30$ arba $n = 60$. Šią išvadą patvirtina tyrimų su įvairiomis uždavinių klasėmis rezultatai. Kai užduočių skaičius $60 < n \leq 120$, mutacijos tikimybę reikia mažinti. Geriausi rezultatai pasiekiami, kai $P_{mut} = 0{,}03$.

5. Sukurtųjų algoritmų testavimo rezultatai palyginti su kitų žinomų metaeuristikos algoritmų taikymo tiems patiems uždaviniams spręsti rezultatais. Tai parodė, jog sukurtieji algoritmai leidžia efektyviai spręsti RITSU uždavinius ir kartais leidžia gauti geresnių rezultatų.
6. Sukurtojo genetinio algoritmo realizacijos taikymas HP montažinių plokščių surinkimo uždaviniui spręsti parodė, jog šis algoritmas gali būti taikomas praktiniams tvarkaraščių uždaviniams spręsti.
7. Sprendžiant įvairias uždavinių klases, sukurtieji algoritmai savo pranašumą parodė padidinus iteracijų skaičių. Atskirų praktinių uždavinių atveju, kai nereikia statistiškai palyginti didelio kiekio uždavinių sprendinių, iteracijų skaičių galima dar padidinti, tuo siekiant rasti sprendinius kiek galima artimesnius optimumui.

Literatūra

1. **Ahuja P., Clark D.W., Rogers A., (1995).** The Performance Impact of Incomplete Bypassing in Processor Pipelines, Proceedings of 28th IEEE/ACM Annual Int. Symp. on Micro-Arch., Ann Arbor, p. 36-45.
2. **Ahuja R.K., Orlin J.B., Sharma D., (2000).** Very large-scale neigborhood search, International transactions in operational research, Vol.7, p. 301-317.
3. **Aldowaisan T., Allahverdi A., (2003).** New heuristics for no-wait flowshops to minimize makespan, Computers & Operations Research, Vol.30, p. 1219-1231.
4. **Alvarez-Valdes R., Crespo E., Tamarit J.M., (2002).** Design and implementation of course scheduling system using tabu search, European journal of operational research, Vol. 137, p. 512-523.
5. **Artiouchine K., Baptiste P., (2005).** Inter-Distance Constraint: An Extension of the All-Different Constraint for Scheduling Equal Length Jobs, Principles and Practice of Constraint Programming, LNCS 3709 Springer -Verlag P. van Beek, p. 62–76.
6. **Azarmi N., Smith R., (1995).** Intelligent scheduling and planning systems for telecommunications resource management, BT technology journal, Advanced information processing techniques for resource scheduling and planning, Vol. 13, No.1, p. 7-15.
7. **aSc TimeTables - School scheduling software for primary/secondary schools, (2007).** Prieiga per internetą <http://www.asctimetables.com/>
8. **Baar T., Brucker P., Knust S., (1999).** Tabu-search algorithms and lower bounds for the resource constrained project scheduling problem. In Voss et al. [Voss S., Martello S., Osman I., Roucairol C., (1999)], p.1–8.
9. **Badri M.A., Davis D.L., Davis D.F., Hollingsworth J., (1998).** A multi-objective course scheduling model: combining faculty preferences for courses and times, Computers & Operations Research, Vol.25, No.4, p. 303-316.
10. **Baker J. E., (1985).** Adaptive selection methods for genetic algorithms, In Grefenstette, J. J. (Ed.), Proceedings of an International Conference on Genetic Algorithms and Their Applications, Hillsdale, N J: Lawrence Erlbaum Associates, p. 101-111.
11. **Bardadym V.A., (1996).** Computer-Aided School and University Timetabling: The New Wave, Selected and Revised Papers of the 1st International Conference on Practice and Theory of Automated Timetabling, (PATAT 1995), Edinburgh, Springer LNCS 1153, p. 22-45.
12. **Baskas A., (1973).** Декомпозиционный метод решения некоторых задач сетевого планирования, Диссертация на соискание учёной степени кандидата технических

наук. Специальность № 05.13.01 – техн. кибернетика и теория информации, Академия Наук Литовской ССР, Институт физики и математики, Вильнюс.

13. **Bast H., (1998).** Dynamic scheduling with incomplete information, Proceedings of the tenth annual ACM symposium on Parallel algorithms and architectures, p. 182-191.
14. **Bataiti R., Tecchiolli G., (1994).** The reactive tabu search, ORSA Journal on Computing, t.6, p. 126–140.
15. **Becker M., Smith S., (1997).** An Ontology for Multi-Modal Transportation Planning and Scheduling, tech. report CMU-RI-TR-98-15, Robotics Institute, Carnegie Mellon University.
16. **Berge C., (1973).** Graphs and Hypergraphs. New York: Elsevier.
17. **Bilkay O., Anlagan O., Kilic S.E., (2004).** Job shop scheduling using fuzzy logic, The International Journal of Advanced Manufacturing Technology, Vol. 23, Numbers 7-8, p. 606-619.
18. **Blazewic J., Ecker K.H., Pesch E., Schmidt G., Weglarz J., (2001).** Scheduling Computer and Manufacturing Processes. Second Editon, Springer-Verlag, Berlin Heidelberg New York. ISBN 3-540-41931-4.
19. **Bouleimen K., Lecocq H., (1998).** A new efficient simulated annealing algorithm for the resource constrained project scheduling problem. Technical report, Universit´e de Li`ege, Belgium.
20. **Bouleimen K., Lecocq H., (2003).** A new efficient simulated annealing algorithm for the resource –constrained project scheduling problem and its multiple mode version, European Journal of Operational Research, Vol.149, p. 268-281.
21. **Brucker P., Knust S., Schoo A., Thiele O., (1998).** A branch and bound algorithm for the resource-constrained project scheduling problem, European Journal of Operational Research, Vol. 107, No. 2, p. 272-288.
22. **Brucker P., Schlie R., (1990).** Job-shop scheduling with multi-purpose machines, Computing, No.45, p. 369-375.
23. **Burke E.K., Jackson K.S., Kingston J.H., Weare R.F., (1997).** Automated Timetabling: The State of the Art, The Computer Journal, Vol. 40, No. 9, p. 565-571.
24. **Carter M.W., Laporte G., (1998).** Recent Development in Practical Course Timetabling, Selected and Revised Papers of the 2nd International Conference on Practice and Theory of Automated Timetabling, (PATAT 1997), Toronto, Springer LNCS 1408, p. 3-19.
25. **Cavalieri S., Mirabella O., (1996).** Neural networks for process scheduling in real-time communication systems, IEEE transactions on neural networks, vol. 7, No.5, p. 1272-1285.
26. **Ceder A., Golany B., Tal O., (2001).** Creating Bus Timetables with Maximal Synchronization, Transportation Research, Vol. 35A, No. 10, p. 913-928.

27. **Chakrapani J., Skorin-Kapov J., (1993).** Connection Machine Implementation of a Tabu Search Algorithm for the Traveling Salesman Problem, Journal of Computing and Information Technology Vol.1(1), p. 29-36.
28. **Chelouah R., Siarry P., (2000).** Tabu search applied to global optimization, European Journal of operational research, vol. 123, p. 256-270.
29. **Chelouah R., Siarry P., (2003).** Genetic and Nelder-Mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions, European Journal of operational research, vol. 148, p. 335-348.
30. **Chu S., Roddick J., Pan J., (2004).** Ant colony system with communication strategies, Information Sciences: an International Journal, 167(1-4), p. 63-76.
31. **Clerc M., (2004).** Discrete Particle Swarm Optimization, New Optimization Techniques in Engineering Springer-Verlag.
32. **Coffman E., Jr., (1982).** An introduction to proof techniques for packing and sequencing algorithms, in Deterministic and Stochastic Scheduling, M. Dempster, et al., Reidel, Amsterdam, p. 245-270.
33. **Colorni A., Dorigo M., Maniezzo V., (1991).** Distributed optimization by ant colonies, in: F. Varela, P. Bourgine (Eds.), First Eur. Conference Artificial Life, p. 134-142.
34. **Connolly D.T., (1990).** An Improved annealing scheme for the QAP, European Journal of Operational Research, Vol. 46, p. 93-100.
35. **De Gloria A., Faraboschi P., Olivieri M., (1992).** A non-deterministic scheduler for a software pipelining compiler, ACM SIGMICRO Newsletter 23(1-2), p. 41-44.
36. **Dell'Amico M., Iori M., Martello S., (2004).** Heuristic Algorithms and Scatter Search for the Cardinality Constrained P||Cmax Problem, Journal of Heuristics, No.10, p. 169–204.
37. **Dorea C.C.Y., (1997).** On the efficiency of a continuous version of the simulated annealing algorithm, Statistics & probability letters, Vol.31, p. 247-253.
38. **Dorigo J.M., Gambardella L.M., (1997).** Ant colony system: a cooperative learning approach to the traveling salesman problem, IEEE Transactions on Evolutionary Computation 1(1), 53-66.
39. **Duncan T., (1990).** Scheduling Problems and Constraint Logic Programming: A Simple Example and its Solution, Technical Report AIAI-TR-120, Artificial Intelligence Applications Institute, University of Edinburgh.
40. **Dzemyda, G., Senkienė E., (1997).** Convergence of parameter clustering based on the simulated annealing, Informatica, Vol. 8, No. 4, p. 465-476, ISSN 0868-4952.
41. **Eisenberg R., (2002).** Solving the generalized resource-constrained project scheduling problem with the heuristic optimization framework HotFrame, Diploma thesis, Technical University Braunschweig.

42. **Eberhart R. C., Kennedy J., (1995).** A new optimizer using particle swarm theory, Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan. pp. 39-43.
43. **Felinskas G., Sakalauskas L., (2003).** Pareto tipo modeliai modeliuojamojo atkaitinimo algoritmuose, Lietuvos matematikos rinkinys, 2003, T.43, spec. nr., p. 573-578, ISSN 0132-2818.
44. **Felinskas G., Sakalauskas L., (2005a).** Tvarkaraščių su ribotais ištekliais sudarymo algoritmai ir jų tyrimas, Matematika ir matematinis modeliavimas, Nr.1, p. 105-110, ISSN 1822-2757.
45. **Felinskas G., Sakalauskas L., (2005b).** Tvarkaraščių su ribotais resursais sudarymo algoritmai ir jų taikymas, Informacinės technologijos '2005. Konferencijos pranešimų medžiaga, Kaunas, Technologija, p. 406-414, ISBN 9955-09-789-2.
46. **Felinskas G., Sakalauskas L., (2006a).** Optimization of resource constrained project schedules by simulated annealing and variable neighborhood search, Technological and economic development of economy, XII tomas, Nr.4., p. 307-313, ISSN 1392-8619.
47. **Felinskas G., Sakalauskas L., (2006b).** Optimization of resource constrained project schedules by genetic algorithm based on the job priority list, Information technology and control, 35 tomas, Nr.4, 412-418, ISSN 1392-124X.
48. **Fink A., Voss St., (2003).** Solving the continuous flow-shop scheduling problem by metaheuristics, European Journal of Operational Research, Vol.151. p. 400-414.
49. **Fores S., Proll L., Wren A., (2002).** TRACS II: a hybrid IP/heuristic driver scheduling system for public transport, Journal of the Operational Research Society, No. 53, p. 1093–1100.
50. **Frostig E., Adiri I., (1985).** Stochastic flowshop no-wait scheduling, J. Appl.Probab. 22, p. 240-246.
51. **Garey M.R., Johnson D.S., (1979).** Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York.
52. **Gelfand S.B., Mitter S.K., (1993).** Metropolis-type annealing algorithms for global optimization in R^d, Control and optimization, Vol.31, No.1, p. 111-131.
53. **Glazebrook K.D., (1979).** Scheduling Stochastic Tasks with Exponential Service Times on Parallel Processors, Journal of Applied Probability, No.16, p. 685-689.
54. **Glazman I.M., Novikov V.G., (1966).** Основы сетевого планирования и управления, Издательство Харьковского университета, Харьков.
55. **Glibovec N.N., Medvidj S.A., (2003).** Генетические алгоритмы и их использование для решения задачи составления расписания, Кибернетика и системный анализ, No.1, p. 95-108.

56. **Glover F., (1990).** Tabu search: part II, ORSA Journal on Computing, t.2, p. 4–32.
57. **Glover, F., (1998).** A Template for Scatter Search and Path Relinking. In: Hao, J.K., Lutton, E., Ronald, E., Schoenauer, M., Snyers D. (Eds.), Artificial Evolution, Lecture Notes in Computer Science 1363, Springer, p. 13-54.
58. **Glover F., Laguna M., (1997).** Tabu search, Boston: Kluwer Acad. Publ.
59. **Glover F., Laguna M., Martí R., (2003).** Scatter Search, Advances in Evolutionary Computation: Theory and Applications, A. Ghosh and S. Tsutsui (Eds.), Springer-Verlag, New York, p. 519-537.
60. **Goldberg D.E., (1989).** Genetic Algorithm in Search, Optimization and Machine Learning, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts.
61. **Gonzalez M., (1977).** Deterministic Processor Scheduling, ACM Comp. Surveys 9(3), p. 173-204.
62. **Graham R.L., (1969).** Bounds on multiprocessing timing anomalies, SIAM Journal of Applied Mathematics, 17, p. 263-269.
63. **Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., (1979).** Optimization and Approximation in Deterministic Sequencing and Scheduling: A survey, Annals of Discrete Mathematics 5, 287–326.
64. **Gulati S., Iyengar S.S., Toomarian N., Protopopescu V., Barhen J., (1987).** Nonlinear Neural Networks for Deterministic Scheduling**,** IEEE International Conference on Neural Networks, Vol. 2, p. 745-752.
65. **Günther H-O., Kim K-H., (2006).** Container terminals and terminal operations, OR Spectrum, Springer Berlin/Heidelberg, Vol. 28, No. 4, p. 437-445.
66. **Hall L.A., (1995).** Approximability of flow shop scheduling, Proc. 36th IEEE Annual Symp. on Foundations of Computer Science, p. 82–91.
67. **Hanafi S., Freville A., (1998).** An efficient tabu search approach for the 0-1 multidimensional knapsack problem, European Journal of Operational Research, t.106, p. 93−100.
68. **Handfield R.B., Nochols E.L. Jr., (1999).** Introduction to Supply Chain Management. Prentice Hall.
69. **Hansen P., Mladenovic N., (1999).** An introduction to variable neighborhood search, Metaheuristics: Advances and trends in local search paradigms for optimization, Kluwer, p. 433-458.
70. **Hartmann S., (1998).** A competitive genetic algorithm for resource-constrained project scheduling. Naval Research Logistics, 45, p. 733–750.
71. **Hartmann S., (2002).** A self-adapting genetic algorithm for project scheduling under resource constraints, Naval Research Logistics, Vol. 49, p. 433-448.

72. **Hartmann S., Kolisch R., (2000).** Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem, European Journal of Operational Research, Vol.127, p. 394-407.
73. **Hemani A., Postula A., (1990).** Scheduling by Self Organization**,** International Joint Conference on Neural Networks,Vol. 2, p. 543-546.
74. **Hendrickson Ch., (2000).** Project Management for Construction. Fundamental Concepts for Owners, Engineers, Architects and Builders. Second Editon. Prieiga per internetą <http://www.ce.cmu.edu/pmbook/>
75. **Herrmann J.W., Lee C.-Y., Snowdon J.L., (1993).** A classification of static scheduling problems, Complexity in Numerical Optimization, p. 203-253.
76. **Hillier M.S., Brandeau M.L., (1998).** Optimal Component Assignment and Board Grouping in Printed Circuit Board Manufacturing, Operations Research 46(5), p. 675–689.
77. **Holland J. H., (1975).** Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. Ann Arbor, MI: Univ. of Michigan Press.
78. **Hoos H.H., Stutzle Th., (2004).** Stochastic Local Search. Foundations and Applications, Morgan Kaufmann / Elsevier.
79. **Ingber L., (1989).** Very fast simulated re-annealing, Mathematical and Computer Modelling, 12(8), p. 967-973.
80. **Iyer S.K., Saxena B., (2004).** Improved genetic algorithm for the permutation flowshop scheduling problem, Computers and operations research, Vol. 31, p. 593-606.
81. **Yamada T., (2003).** Studies on Metaheuristics for Jobshop and Flowshop Scheduling Problems, Doctoral thesis, Kyoto University, Japan.
82. **Yang B., Geunes J., O'Brien W. J., (2001).** Resource-Constrained Project Scheduling: Past Work and New Directions, Research Report 2001-6, Department of Industrial and Systems Engineering, University of Florida.
83. **Yang R.L., (2000).** Convergence of the Simulated Annealing Algorithm for Continuous Global Optimization, Journal of optimization theory and applications, Vol. 104, No. 3, p. 691-716.
84. **Yoshikawa M., Kaneko K., Yamanouchi T., Watanabe M. (1996).** A Constraint-Based High School Scheduling System, IEEE Expert: Intelligent Systems and Their Applications 11(1), p. 63-72.
85. **Jozefowska J., Mika M., Rozycki R., Waligora G., Weglarz J., (1998).** Local search metaheuristics for discrete-continuous scheduling problems, European Journal of Operational Research, Vol.107, p. 354-370.

86. **Kalanta St., (2003).** Taikomosios optimizacijos pagrindai. Tiesinių uždavinių formulavimas ir sprendimo metodai, Vilnius, „Technika“.
87. **Kelley J.E., Jr., Walker M.R., (1959).** Critical-path planning and scheduling, Proceedings of Eastern Joint Computer Conference, Boston MA, p. 160-173.
88. **Kelley J.E., Jr., (1961).** Critical-Path Planning and Scheduling: Mathematical Basis, Operations Research, Vol.9, No.3, p. 296-320.
89. **Khouja M., Michalewicz Z., Wilmot M., (1998).** The use of genetic algorithms to solve the economic lot size scheduling problem, European journal of oper. research, Vol. 110, p. 509-524.
90. **Kim K.H., Moon K.C., (2003).** Berth scheduling by simulated annealing, Transportation research, Part B Vol. 37, p. 541-560.
91. **Kirkpatrick S., Gelatt Jr. C.D., Vecchi M.P., (1983).** Optimization by simulated annealing, Science, 220, p. 671-680.
92. **Klein R., (2000).** Scheduling of Resource-Constrained Project, Kluwer Academic Publisher, Boston/Dordrecht/London.
93. **Kocetov J.A., Stoliar A.A., (2003).** Использование чередующихся окрестностей для приближенного решения задачи календорного планирования с ограниченными ресурсами, Дискретный анализ и иследование операций, сер. 2, vol. 10, No 2, p. 29-55.
94. **Kolisch R., (1996a).** Efficient priority rules for the resource-constrained project scheduling problem, Journal of Operations Management, Nr.14, p.179-192.
95. **Kolisch R., (1996b).** Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation, European Journal of Operational Research, Vol. 90, Nr.2, p. 320-333.
96. **Kolisch R., Drexl A., (1996).** Adaptive search for solving hard project scheduling problems. Naval Research Logistics, 43, p. 23–40.
97. **Kolisch R., Hartmann S., (1999).** Project scheduling: Recent models, algorithms and applications, Kluwer, Amsterdam, p. 147-178.
98. **Kolisch R., Hartmann S., (2005).** Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update, European Journal of Operational Research, preprint.
99. **Kolisch R., Sprecher A., (1996).** PSPLIB - A project scheduling library, European Journal of Operational Research, Vol. 96, p. 205-216.
100. **Lageweg B.J., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., (1982).** Computer-aided complexity classification of combinatorial problems, Comm. ACM 25, 817-822.
101. **Laguna M., homepage, (2006).** Prieiga per internetą <http://leeds-faculty.colorado.edu/laguna>

102. **Laplagne I., Kwan R.S.K., Kwan A.S.K., (2005).** A hybridised integer programming and local search method for robust train driver schedules planning. Lecture Notes in Computer Science, 3616, p. 71–85.

103. **Lassaigne R., Rougemont M., (1999).** Logika ir algoritmų sudėtingumas, Žara, Vilnius.

104. **Lawler E. L., Sahni S., (1990).** Optimal Preemptive Scheduling of Two Unrelated Processors, ORSA Journal on Computing, Vol. 2, No. 3, p. 219-224.

105. **Leon V. J., Ramamoorthy B., (1995).** Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling. OR Spektrum, 17, p. 173–182.

106. **Lesaint D., Voudouris Chr., Azarmi N., (2000).** Dynamic Workforce Scheduling for British Telecommunications plc, ACM Portal, Volume 30, Issue 1, p. 45 – 56.

107. **Liu J., Maccarthy B. L., (1996)**. The classification of FMS scheduling problems, International journal of production research, vol. 34, No3, pp. 647-656.

108. **Locatelly M., (2000a).** Convergence of Simulated Annealing algorithm for continuous global optimization, Journal of global optimization, Vol. 18, p. 219-234.

109. **Locatelly M., (2000b).** Simulated Annealing algorithms for continuous global optimization: convergence conditions, Journal of optimization theory and applications, Vol. 104, No.1, p. 121-133.

110. **Luh P. B., Zhao X., Thakur L. S., Chen K. H., Chieu T., Chang S., (1999).** Architectural design of neural network hardware for job shop scheduling, CIRP Annals, Vol. 48, No. 1, p. 373-376.

111. **Lundy M., Mees A., (1986).** Convergence of an annealing algorithm, Mathematical Programming, Vol. 34, p. 111-124.

112. **Machado J.M., Shiyou Y., Ho S.L., Peihong N., (2001).** A common Tabu search algorithm for the global optimization of engineering problems, Computer methods in applied mechanins and engineering, Vol.190, p. 3501-3510.

113. **MathSoft WinQSB, (2003).** Prieiga per internetą <http://www.msmiami.com/product.cfm?ProductID=157>

114. **Metropolis N., Rosenbluth A.W., Rosenbluth M.N., Teller A.H., Teller E. (1953).** Equation of State Calculation by Fast Computing Machines, J. of Chem. Phys., 21, p. 1087-1091.

115. **Mimosa Scheduling Software, (2007).** Prieiga per internetą <http://www.mimosasoftware.com/>.

116. **Minga A.K., (1986).** Genetic algorithms in aerospace design, The AIAA Southeastern Regional Student Conference, Huntsville, AL.

117. **Misevičius A., (2000).** A new improved simulated annealing algorithm for the quadratic assignment problem, Informacinės technologijos ir valdymas, Nr. 4(17), ISSN 1392-124X.

118. **Misevičius A., Lenkevičius A., (2003).** A modification of Tabu Search and its application to the quadratic assignment problem, Informacinės technologijos ir valdymas, Nr. 2(27), ISSN 1392-124X.

119. **Mockus J., (2000).** A set of examples of global and discrete optimization. Prieiga per internetą <http://soften.ktu.lt/~mockus>

120. **Möhring R., Schulz A., Uetz M., (1999).** Approximation in stochastic scheduling: the power of LP-based priority policies, Journal of the ACM 46(6), p. 924-942.

121. **Moscato P. (1989).** On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms , Caltech Concurrent Computation Program, C3P Report 826.

122. **Nemeth G., Lovrek I., Sinkovic V., (1997).** Scheduling problems in parallel systems for telecommunications, Computing, Vol. 58, No.3, p. 199-223.

123. **Onbasoglu E., Ozdamar L., (2001).** Parallel simulated annealing algorithms in global optimization, Journal of global optimization, Vol.19, p. 27-50.

124. **Open Source - Job Scheduler, (2005).** Prieiga per internetą <http://www.sos-berlin.com/modules/cjaycontent/index.php?id=osource_scheduler_introduction_en.htm >

125. **Parker R., (1995).** Deterministic Scheduling, Chapman-Hall.

126. **Petrovic S., Burke E., (2004).** University Timetabling, Handbook of Scheduling: Algorithms, Models, and Performance Analysis, CRC Press, Chapter 45, p. 1-23.

127. **„Pikas“ - maršrutinio transporto darbo planavimas, (2007).** Prieiga per internetą <http://www.merakas.lt/lt/pikas/>

128. **Pinedo M., (2005).** Planning and Scheduling in Manufacturing and Services, Springer Series in Operations Research and Financial Engineering, Chapters 1 and 2.

129. **Project Standard 2003 apžvalga, (2003).** Prieiga per internetą **<**http://www.microsoft.com/lietuva/office/project/prodinfo/standard/overview.mspx>

130. **PSPLIB - A project scheduling library, (1996).** Prieiga per internetą <http://www.bwl.uni-kiel.de/Prod/psplib/>

131. **Pušinaitis J., (2004).** Statybos darbų organizavimo pagrindai. Prieiga per internetą <http://www.pusinaitis.lt/>

132. **Radcliffe N.J., Surry P.D., (1994).** Formal memetic algorithms, Evolutionary Computing: AISB Workshop, Ed: T. Fogarty, Springer-Verlag.

133. **Reklaitis G.V., (1982).** Review of scheduling of process operations, AIChE Symposium Series, 78, 119-133.

134. **ResSched – Scheduling Software and Management Tools, (2006).** Prieiga per internetą <http://www.madrigalsoft.com/fresmulti.html >

135. **Sakalauskas L., (2002).** Non-linear stochastic programming by Monte-Carlo estimators, European journal on operational research, Vol. 137, p. 558-573.
136. **Sakalauskas L., Bartkutė V., Felinskas G., (2006).** Optimality testing in stochastic and heuristic algorithms, Technological and economic development of economy, t. XII, Nr.1, p. 4-10, ISSN 1392-8619.
137. **Sakalauskas L., Felinskas G., (2006).** Tvarkaraščių su ribotais ištekliais sudarymo euristiniai algoritmai, jų tyrimas bei taikymai, Informacijos mokslai, 38 tomas, p. 90-103, ISSN 1392-1487.
138. **Savage C., (1997).** A Survey of Combinatorial Gray Codes, Society of Industrial and Applied Mathematics Review 39, p. 605-629.
139. **Schirmer A., (2000).** Case-based reasoning and improved adaptive search for project scheduling. Naval Research Logistics, 47, p. 201–222.
140. **Schoen F., (1993).** A wide class of test functions for global optimization, Journal of global optimization, Vol.3, p 133-137.
141. **School–Complete.** Prieiga per internetą < http://soften.ktu.lt/~mockus/>
142. **Schopf J., Berman F., (1999).** Stochastic scheduling, ACM Press New York, NY, USA.
143. **Schroth G., (1997).** Fundamentals of School Scheduling, Technomic Publishing Co.
144. **Sevastjanov S., Woeginger G., (1998).** Makespan minimization in open shops: A polynomial time approximation scheme, Mathematical Programming, 82(1-2).
145. **Shade J.J., Orman A.J., (1997).** Scheduling installations in the telecommunications industry, European Journal of Operational Research, Vol. 102, No. 1, p. 73-87.
146. **Shapiro J.F., (2001).** Modeling the Supply Chain. Duxbury Thomson Learning.
147. **Shmoys D.B., Stein C., Wein J., (1994).** Improved approximation algorithms for shop scheduling problems, SIAM Journal of Computing, No.23, p. 617–632.
148. **Shrivastava A., Earlie E., Dutt N., Nicolau A., (2004).** Operation tables for scheduling in the presence of incomplete bypassing, Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, p. 194-199.
149. **Simulation with ARENA, (2007).** Prieiga per internetą <http://www.arenasimulation.com/>
150. **Skorin-Kapov J., (1990).** Tabu Search Applied to the Quadratic Assignment Problem, ORSA Journal on Computing, Vol.2(1), p 33-45.
151. **Smykalov P.J., (2001).** РЕКТОР - программа составления расписания уроков. Prieiga per internetą <http://www.mnogosmenka.ru/drugoe/rektor.htm>
152. **Solodovnikova O.S., Solodovnikov S.V., (2002).** Информационные технологии в задачах составления расписания в вузах. Prieiga per internetą <http://www.mnogosmenka.ru/drugoe/telematika.htm>

153. **Steenken D., Voss St., Stahlbock R., (2004).** Container terminal operation and operations research - a classification and literature review, OR Spectrum, Vol. 26, p. 3-49.

154. **Supply Chain Management Research Center (2006).** Prieiga per internetą <http://www.cio.com/research/scm/edit/012202_scm.html>

155. **Sviridenko M., Jansen K., Solis-Oba R., (1999).** Makespan minimization in job-shops: A polynomial time approxmiation scheme, Proc. 31st Annual ACM Symp. on Theory of Computing, p. 394–399.

156. **Taillard E.D., (1991).** Robust taboo search for the quadratic assignment problem. Parallel Computing, 17, p. 443-455.

157. **Tanajev V.S., Kovaliov M.J., Šafranskij J.M., (1998).** Теория расписаний. Групповые технологии, Минск, Институт технической кибернетики, ISBN 985-6453-20-8.

158. **Tandon M., Cummings P.T., La Van M.D., (1991).** Flowshop sequencing with non-permutation schedules, Computers and Chemical Engineering, No.15, p. 601-607.

159. **Tannenbaumas P., Arnoldas R., (1995).** Kelionės į šiuolaikinę matematiką,Vilnius, TEV.

160. **Vaessens R.J.M., (1995).** Generalized Job Shop Scheduling: Complexity and Local Search, Ph.D. thesis, Eindhoven University of Technology.

161. **Vaičiulis B., Matulis V., (1970).** О кодировании и машинном расчёте временных характеристик больших сетей, Научные труды высших учебных заведений Литовской ССР, Автоматика и вычислительная техника, III.

162. **Vaičiulis B., (1973).** Комплекс машинно-ориентированных алгоритмов решения сетевых задач, Диссертация на соискание учёной степени кандидата технических наук. Специальность № 05.13.01 – техн. кибернетика и теория информации, Академия Наук Литовской ССР, Институт физики и математики, Вильнюс.

163. **Viliūnas P., (1976).** Tinklinio grafiko sudarymo metodika, Kaunas, KPI.

164. **Voss St. (1992).** Network Design Formulation in Schedule Synchronization' In Computer-Aided Transit Scheduling, Desrochers and Roussea (Eds.), Springer Verlag Berlin-Hiedelberg, p. 137-152.

165. **Voss St., (2001).** Meta-heuristics: The State of the Art, Local Search for Planning and Scheduling, LNAI 2148, p. 1-23.

166. **Voss S., Martello S., Osman I., Roucairol C., (1999).** Meta-heuristics: Advances and trends in local search paradigms for optimization. Kluwer Academic Publishers.

167. **Wang L., Li D.W., (2002).** A scheduling algorithm for flexible flow shop problem, IEEE Proceedings of the 4th World Congress on Intelligent Control and Automation, Vol.4, p. 3106-3108.

168. **Wang Y.Z., (2003).** Using genetic algorithm methods to solve course scheduling problems, Expert systems with applications, Vol. 25, p. 39-50.

169. **Wittrock R.J. (1988).** An adaptive scheduling algorithms for flexible flow lines, Operations Research, Vol. 33, No.4, p. 445-453.

170. **Zhang L., (2005).** Application of Particle Swarm Optimization on Batch Process Scheduling, Proceedings of the 43rd ACM Southeast Conference, Volume 1, p. 155 – 156

171. **Žilinskas A., (1986).** Глобальная оптимизация, Vilnius, Mokslas.

172. **Žilinskas A.**, **Zhigliavsky A., (1991).** Методы поиска глобального экстремума, Maskva, Nauka.

Priedai

A priedas. Grafo apibrėžimas bei pagrindinės sąvokos

A.1 Grafo ir orgrafo apibrėžimas

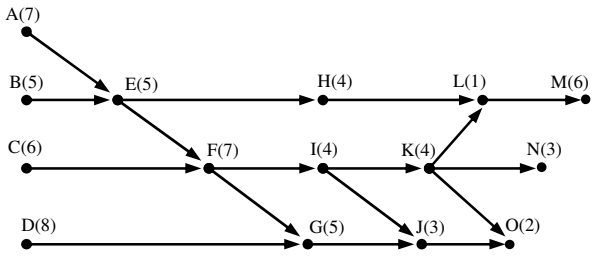
Tvarkaraščių duomenims formalizuoti yra taikomos grafų teorijos sąvokos. Tarkime, V – netuščia viršūnių aibė, E – visų aibės V galimų dvielemenčių poaibių aibė (visų įmanomų briaunų aibė), U – bet koks aibės E poaibis (briaunų aibė). Grafu vadinama pora (V, U), t.y. G =(V, U).

Grafai, kuriuose briaunos turi kryptis, vadinami kryptiniais (orientuotaisiais) grafais arba orgrafais. Šiuo atveju briaunos vadinamos lankais.

Lankai (briaunos) vadinami gretimais, jei jie turi bendrą viršūnę. Viršūnės yra gretimos, jei jas jungia vienas lankas (briauna). Yra sakoma, kad viršūnės u ir v incidentiškos lankui (briaunai) (u, v), jei tarp šių viršūnių yra tokia briauna.

Grafo grandine vadinama briaunų (lankų) seka (v_0, v_1), (v_1, v_2), ..., (v_{n-1}, v_n). Norint apibrėžti grandinę, pakanka eilės tvarka nurodyti viršūnes, per kurias eina grandinė. Grandinės, kurių pirma viršūnė sutampa su paskutine, vadinamos ciklais. Grandinė (ciklas) yra paprastoji, jei ta pati viršūnė grandinėje pasikartoja tik vieną kartą (ciklo atveju pirma viršūnė gali sutapti su paskutine). Grandinės ilgis – briaunų, priklausančių grandinei, skaičius. Yra žinoma, jog grafas su n viršūnių gali turėti ne daugiau, kaip (n-1)! ciklų (Berge, 1973).

Kelias – tai grandinė kryptiniame grafe, o kontūras – ciklas kryptiniame grafe. Kelias iš viršūnės *X* į viršūnę *Y* orgrafe yra kryptinių lankų seka, prasidedanti *X* ir pasibaigianti *Y*. Pavyzdžiui, kelias iš viršūnės A į M yra grandinė, kurią sudaro lankų seka (A,E), (E, H), (H, L), (L,M). (A.1 pav.)



A.1 pav. Užduočių nuoseklumo sąryšius vaizduojantis grafas

A.2 Gretimumo matricos

Grafai formaliai aprašomi gretimumo matricomis. Gretimumo matrica – tokia matrica, kurioje grafo viršūnes atitinka stulpelių ir eilučių pavadinimai, o matricos elementai įgyja „teisingą“ (true, 1) arba „klaidingą“ (false, 0) reikšmę priklausomai nuo to ar viršūnės (eilutės ir stulpelio pavadinimuose) susietos briauna. Paprastai priimama, jog pati su savimi viršūnė briaunos neturi,

todėl tokios lentelės diagonalė užpildyta reikšmėmis „klaidinga“. Nekryptinio grafo gretimumo matrica yra simetriška. Tokia matrica paprasčiausiai realizuojama dvimačiu masyvu. Jeigu grafas turi nedaug briaunų, tai jį patogiau aprašyti gretimumo sąrašais. Kiekvienai grafo viršūnei sukuriama po sąrašą. Juose saugomi kiekvienai viršūnei gretimų viršūnių pavadinimai arba numeriai.

Tvarkaraščio darbų eiliškumas yra aprašomas darbų nuoseklumo matrica. Ji yra gaunama darbus pavaizdavus orgrafo viršūnėmis, kurios yra sujungiamos, jei tarp atitinkamų darbų egzistuoja nuoseklumo sąryšis.

B priedas. Metrinės ir topologinės erdvės

Metrinė erdvė – tai tokia aibė, kurioje įvestas atstumas tarp elementų. Metrine erdve vadinama pora (X, ρ), sudaryta iš tam tikros aibės (erdvės) X elementų (taškų) ir atstumo (metrikos), t.y. vienareikšmės, neneigiamos realios funkcijos ρ(x,y), apibrėžtos bet kuriems x ir y iš aibės X ir tenkinančios aksiomas:

1) ρ(x,y)=0 tada ir tik tada, kai x=y.

2) (simetrijos savybė) ρ(x,y)= ρ(y,x).

3) (trikampio savybė) ρ(x,z)≤ ρ(x,y) + ρ(y,z).

Pačią metrinę erdvę, t.y. porą (X, ρ) žymime R=(X, ρ). Atviruoju rutuliu $B(x_0,r)$ metrinėje erdvėje R vadinsime taškų $x\in R$ aibę, tenkinančią $\rho(x,x_0)<r$ (x_0 – centras, r – spindulys). Uždarasis rutulys – kai $\rho(x,x_0)\le r$. Atvirąjį rutulį spinduliu ε su centru x_0 mes dar vadinsime taško x_0 ε-aplinka ($O_\varepsilon(x_0)$).

Topologinės erdvės yra įvedamos apibrėžiant atvirųjų poaibių sistemą duotoje aibėje. Tegul X – tam tikra aibė. Topologija aibėje X vadinama bet kokia sistema τ jos poaibių G, tenkinanti tokius reikalavimus:

1) Pati aibė X ir tuščia aibė Ø priklauso τ.

2) Sąjunga $\bigcup_\alpha G_\alpha$ bet kokio skaičiaus (baigtinio ir begalinio) ir sankirta $\bigcap_{k=1}^{n} G_k$ bet kokio baigtinio skaičiaus aibių iš τ priklauso τ.

Aibė X su duota joje topologija τ, t.y. pora (X, τ), vadinama topologine erdve. Aibės, priklausančios sistemai τ, vadinamos atvirosiomis. Nusakyti topologinę erdvę – reiškia nusakyti tam tikrą aibę X ir nusakyti joje topologiją τ, t.y. nurodyti tuos poaibius, kurie laikomi atvirais aibėje X. Aišku, kad toje pačioje aibėje X galima įvesti skirtingas topologijas, tuo pačiu paverčiant aibę X skirtingomis topologinėmis erdvėmis. Dažnai topologinę erdvę, t.y. porą (X, τ), mes žymėsime viena raide T. Topologinės erdvės elementus vadinsime taškais.

Aibės T \ G, papildančios atvirąsias, vadinamos topologinės erdvės T uždarosiomis aibėmis. Iš aksiomų 1) ir 2) (ir dar kai kurių sąryšių) seka:

1. Tuščia aibė Ø ir visa T – uždaros.

2. Bet kokio skaičiaus (baigtinio ir begalinio) uždarųjų aibių sankirta ir baigtinio skaičiaus uždarųjų aibių sąjunga yra uždara.

Šių apibrėžimų pagrindu yra įvedamos kitos topologinių erdvių sąvokos – aplinka, sąlyčio taškas, uždarinys ir t.t. Taško $x\in T$ aplinka vadiname bet kokią atvirąją aibę $G\subset T$, kurioje yra taškas x. Taškas $x\in T$ vadinamas aibės $M\subset T$ sąlyčio (sienos) tašku, jei kiekvienoje taško x aplinkoje yra bent vienas taškas iš aibės M. Taškas x vadinamas ribiniu aibės M tašku, jei kiekvienoje x aplinkoje

yra bent vienas taškas iš aibės M, skirtingas nei x. Aibė visų M sąlyčio (sienos) taškų vadinama aibės M uždariniu ir žymima [M]. Galima įrodyti, jog tik uždarosioms aibėms ir tik joms, apibrėžiamoms kaip atvirųjų aibių papildiniai, galioja [M]=M. Kaip ir metrinės erdvės atveju, [M] yra mažiausia uždara aibė, kurioje yra M. Bet kokia metrinė erdvė kartu yra ir topologinė erdvė.

Tegul toje pačioje aibėje X duotos dvi topologijos τ_1 ir τ_2 (tuo pačiu duotos 2 topologinės erdvės $T_1=(X, \tau_1)$ ir $T_2=(X, \tau_2)$). Mes sakysime, kad topologija τ_1 „stipresnė“ ar „tikslesnė“ už topologiją τ_2 , jei aibių sistema τ_2 yra aibėje τ_1. Tada apie τ_2 sakome, jog ji yra „silpnesnė“ ar „grubesnė“ už τ_1.

Visų galimų aibės X topologijų aibėje įvedama dalinė tvarka (sutvarkyta aibė) – topologija τ_2 yra pirmesnė už τ_1, jei ji yra „silpnesnė“ nei τ_1. Šioje topologijų aibėje yra maksimalus elementas – topologija, kurioje visos aibės yra atviros, ir minimalus elementas – topologija, kurioje atviros yra tik visa aibė X ir tuščia aibė Ø.

Kartais būna patogu nusakyti ne visą topologiją, o tik tam tikrą jos dalį, t.y., tam tikrą atvirųjų aibių poaibį, pagal kurį vienareikšmiškai nusakoma visų atvirųjų poaibių aibė (pvz., taip metrinėje erdvėje pradžioje įvesta atvirojo rutulio (ε aplinkos) sąvoka, o paskui apibrėžtos atvirosios aibės, tokios, kuriose kiekvienas taškas aibėje yra kartu su savo ε aplinka). Kitaip tariant – metrinėje erdvėje atvirosios yra tik tos aibės, kurias galima išreikšti kaip atvirųjų rutulių sumas (baigtines ar begalines). Atskiru atveju, tiesėje atvirosios aibės išreiškiamos intervalų sumomis. Aibė ***G*** atvirųjų poaibių vadinama topologinės erdvės T baze, jei bet kuri atvira aibė erdvėje T gali būti išreikšta, kaip suma (sąjunga) tam tikro skaičiaus (baigtinio ar begalinio) aibių iš ***G***. Taigi, erdvės T topologiją τ galima apibrėžti nurodant šioje erdvėje tam tikrą jos bazę ***G***. Ši topologija τ sutampa su rinkiniu aibių, išreiškiamų, kaip sumos (sąjungos) aibių iš ***G***.

Bet kuri bazė ***G*** topologinėje erdvėje $T=(X, \tau)$ turi šias dvi savybes:

1) Bet kuris taškas $x \in X$ yra bent vienoje $G \in \boldsymbol{G}$.

2) Jei x yra dviejų aibių iš ***G*** – G_1 ir G_2 sankirtoje, tai egzistuoja tokia $G_3 \in \boldsymbol{G}$, kad $x \in G_3 \subset G_1 \cap G_2$.

Iš tikro 1) savybė reiškia, kad visa X, būdama atvirąja, turi būti išreiškiama, kaip suma (sąjunga) kažkokių aibių iš ***G***. 2) savybė išplaukia iš to, kad $G_1 \cap G_2$ yra atviroji ir todėl ji yra suma (sąjunga) kažkokių bazės elementų.

Topologinė erdvė T vadinama metrizuojama, jei jos topologiją galima nusakyti kokia nors metrika.

C priedas. Algoritmų sudėtingumas

Bendru atveju kiekvienam uždaviniui nusakyti yra reikalinga tokia pradinė informacija:

I) Visų uždavinio parametrų sąrašas,

II) Reikalavimų, kuriuos turi tenkinti uždavinio sprendinys, sąrašas.

Keisdami uždavinio parametrus, gausime skirtingus uždavinio variantus arba jų klasę. Pradinės informacijos sąrašo ilgis gali būti naudojamas pradinės informacijos kiekiui apibūdinti. Algoritmu yra vadinamas sąrašas procedūrų, kurias atliekant žingsnis po žingsnio, yra gaunamas uždavinio sprendinys. Tegul yra sprendžiamas uždavinys, kurio pradinės informacijos sąrašo ilgis yra n. Kuriamo algoritmo efektyvumo svarbi charakteristika yra jo sudėtingumas. Algoritmų sudėtingumas gali būti nagrinėjamas kompiuterio laiko, kompiuterio atminties arba programinio kodo sudėtingumo požiūriu, priklausomai nuo pradinės informacijos sąrašo ilgio. Kompiuterio laikas, kurio reikia išspręsti daugeliui kombinatorinių uždavinių, labai sparčiai auga didėjant n, o kompiuterio atminties ištekliai bei kodo sudėtingumas dažniausiai nuo n arba nepriklauso, arba auga nežymiai.

Uždavinio laiko sudėtingumo skaičiavimas vertina kiek laiko reiktų tam tikrai problemai su tam tikru duomenų dydžiu spręsti efektyviausiu algoritmu. Tegul sprendžiamas uždavinys koduojamas informacijos kiekiu lygiu n bitų. Jei problema išsprendžiama per algoritmo žingsnių skaičių, proporcingą n^2, tai sakoma, jog tokia problema yra n^2 sudėtingumo. Iš tiesų, kiekviena algoritmo realizacija spręstų problemą skirtingu žingsnių skaičiumi, bet proporcingu n^2, tai sąlyginai žingsnių skaičius problemai spręsti žymimas $O(n^2)$. Algoritmų sudėtingumui žymėti yra taikomi asimptotiniai žymėjimai. Asimptotinei „viršutinei“ ribai žymėti yra taikomas O žymėjimas, t.y. $f(n) = O(g(n))$, jei egzistuoja konstantos c ir n_0 tokios kad $cg(n) \geq f(n)$ visiems $n \geq n_0$. O dažniausiai naudojamas algoritmo blogiausiam atvejui apibūdinti. Asimptotinei „apatinei“ ribai žymėti yra taikomas Ω žymėjimas: $f(n) = \Omega(g(n))$, jei egzistuoja konstantos c ir n_0 tokios kad $cg(n) \leq f(n)$ visiems $n \geq n_0$. Ω dažniausia apibūdina algoritmo geriausią atvejį arba apatinę ribą. Asimptotinei „ankštai“ ribai žymėti yra taikomas Θ žymėjimas: $f(n)=\Theta(g(n))$ jei egzistuoja konstantos c_1, c_2 ir n_0, tokios kad $c_1 g(n) \leq f(n) \leq c_2 g(n)$ visiems $n \geq n_0$. Šiuo atveju galioja savybė, $f(n)=\Theta(g(n)) \Leftrightarrow f(n)=O(g(n))$ ir $f(n) = \Omega(g(n))$. Asimptotinei „negriežtai viršutinei“ ribai žymėti yra taikomas o žymėjimas: $f(n) = o(g(n))$, jei egzistuoja konstantos $c > 0$ ir $n_0 > 0$, tokios kad $cg(n) > f(n)$ visiems $n \geq n_0$. Asimptotinei „negriežtai apatinė“ ribai žymėti yra taikomas ω žymėjimas: $f(n) = \omega(g(n))$, jei egzistuoja konstantos $c > 0$ ir $n_0 > 0$, tokios kad $cg(n) < f(n)$ visiems $n \geq n_0$.

C.1 lentelė. Dažniausi algoritmų sudėtingumo žymėjimai ir jų pavadinimai

Žymėjimas	Sudėtingumas	Klasė
O(1)	konstantinis	Polinominė (P)
O(log *n*)	logaritminis	
O([log *n*]^c)	polilogaritminis	
O(*n*)	tiesinis	
O(*n* log *n*)	supertiesinis	
O(n^2)	kvadratinis	
O(n^c)	polinominis (geometrinis)	
O(c^n)	eksponentinis	Eksponentinė (NP)
O(*n!*)	faktorialinis	
O(n^n)		

Paprasčiausių algoritmų sudėtingumo pavyzdžiai:

paieškos algoritmas tiesinėje nesurūšiuotų duomenų struktūroje yra tiesinio sudėtingumo – *O(n)*;

paieškos algoritmas tiesinėje surūšiuotų duomenų struktūroje yra logaritminio sudėtingumo – *O(*log *n);*

rūšiavimo algoritmas tiesinėje duomenų struktūroje yra supertiesinio sudėtingumo – *O(n* log *n).*

Kuprinės uždavinys yra eksponentinio sudėtingumo – *O* (c^n).

Kėlinių perrinkimas yra faktorialinio sudėtingumo – *O (n!).*

Uždaviniai, kuriems išspręsti pakanka kompiuterio laiko, augančio polinominiu greičiu, priklauso polinominio sudėtingumo klasei P. Kiti uždaviniai priklauso klasei NP. Pagrindinė algoritmų sudėtingumo teorijos problema yra patvirtinti arba paneigti hipotezę ***P=NP***. Jei ši hipotezė pasitvirtintų, tai būtų galima tikėtis sukurti efektyvius algoritmus plačioms kombinatorinių uždavinių klasėms spręsti. Jeigu klasės P ir NP yra skirtingos, tai tenka kurti specialius algoritmus vis siauresnėms uždavinių klasėms (Garey, Johnson, 1979, Lassaigne, Rougemont, 1999). Atskirą NP poklasį sudaro NP-pilnieji uždaviniai, t.y. tokie uždaviniai, į kuriuos gali būti per polinominį laiką suvedami visi kiti tos klasės uždaviniai. Yra išskirti šeši NP-pilnieji uždaviniai (Garey, Johnson, 1979).

D priedas. PSPLib pradinių duomenų rinkiniai

Bibliotekoje atskirais failais patalpinti tvarkaraščių sudarymo uždavinių pradiniai duomenys bei informacija apie projektą. Projekto su 30 užduočių pradinių duomenų rinkinys, perskaičius visus pradinius duomenis, atrodo taip:

```
************************************************************************
užduočių (įskaitant fiktyvias) :  32
visų užduočių trukmė :  164  IŠTEKLIAI  - atnaujinami  :  4   R
************************************************************************
PROJEKTO INFORMACIJA:
pronr. #užd. prad.laikas reikiamas baig.laikas  vėlav.kaina
   1     30       0              41                 21
************************************************************************
NUOSEKLUMO SĄRYŠIAI:
Užd.Nr. #būsenų  #užd. po jo   po jo atliekamos užduotys (numeriai)
   1        1          3           2   3   4
   2        1          3          10  11  28
  ...
  31        1          1          32
  32        1          0
************************************************************************
SĄNAUDOS/TRUKMĖS:       Reikiami išteklių kiekiai
Užd.Nr. būsena trukmė  R 1  R 2  R 3  R 4
------------------------------------------------------------------------
  1        1     0       0    0    0    0
  2        1     2       1    2    4    0
...
 31        1     3       0    4    5    1
 32        1     0       0    0    0    0
************************************************************************
TURIMAS IŠTEKLIŲ KIEKIS:  R1=24  R2=23  R3=25  R4=33
************************************************************************
```

Duomenys, skirti testavimo programoms nuskaityti, pateikti J3010_1.rcp (iš viso – 480 atskirų atvejų):

```
32      4
24      23      25      33
0       0       0       0       0       3       2       3       4
2       1       2       4       0       3       10      11      28
...
3       0       4       5       1       1       32
0       0       0       0       0       0
```

Iš šių duomenų testavimo programa nuskaito visą šio poskyrio pradžioje pateiktą informaciją.

Optimalūs (ar euristiniai) sprendiniai pateikiami tokioje formoje:

```
================================================================
Atskirų atvejų rinkinys : j30  Tipas: sm
================================================================
Parametrai Atvejai Vykd.laikas CPU-Laikas[sek.]
----------------------------------------------------------------
1             1         43       0.30
1             2         47       0.11
1             3         47       0.12
...
48            8         44       0.01
48            9         59       0.00
48            10        54       0.01
================================================================
```

(iš viso – 480 atvejų)

E priedas. Tvarkaraščių optimizavimo, panaudojant pirmumo sąrašą, programinės įrangos aprašymas

E.1 Sukurtosios funkcijos ir procedūros, kurios naudojamos MA ir GA algoritmų realizacijose.

Sukurtosios funkcijos:

`Leistinas(kiek:integer;x:binmas;y:intvekt):boolean` – patikrina, ar užduočių pirmumo sąrašas Y yra leistinas, atsižvelgiant į visus nuoseklumo sąryšius, nusakytus matrica X, kai užduočių skaičius yra K.

Sukurtosios procedūros:

`PilnSarMatrica(var kiek:integer;var x,y:binmas)` – duotai nuoseklumo sąryšių matricai X sukonstruoja pilnąją sąryšių matricą Y, kai užduočių skaičius KIEK.

`PirmSarGen(var Kiek:integer; var x:binmas; var y:intvekt)` – pagal duotą pilnąją nuoseklumo sąryšių matricą X su KIEK užduočių sunkstruoja leistinąjį pirmumo sąrašą Y.

`PirmSarKeit(var Kiek:integer; var y:intvekt)` – duotajame pirmumo vektoriuje Y su KIEK užduočių atlieka pirmumo sąrašo keitimą, pritaikant vieną iš dviejų elementarių operacijų (kiekviena su tikimybe p=0,5) atsitiktinėms užduotims – užduoties perkėlimą arba dviejų užduočių sukeitimą.

`PirmSarKeit2(var Kiek,qq:integer; var y:intvekt)` – duotajame pirmumo vektoriuje Y su KIEK užduočių atlieka pirmumo sąrašo keitimą, keičiant nurodytą užduotį, pritaikant vieną iš dviejų elementarių operacijų (kiekviena su tikimybe p=0,5) – užduoties perkėlimą arba sukeitimą su kita užduotimi.

`Procedure NDekoder(kiekr,kiekd:integer;sarys:binmas;`
`res:masiv;pirsar:intvekt;truk:realvekt;var start:realvekt;var Tproj:real)` – pagal duotus duomenis (resursų skaičių, užduočių skaičių, nuoseklumo sąryšių matricą, resursus, pirmumo sąrašą, užduočių trukmes, suskaičiuoja užduočių pradžios laikus bei galutinį projekto baigimo laiką, t.y. tikslo funkciją.

`gylio_sk(cc, tt:real; var rro:integer)` – pagal duotą temperatūrą bei kitus parametrus, parenka tam tikrą aplinkos gylį RRO, kuris turi tenkinti kitus programoje nurodytus kriterijus. Šis aplinkos gylis nurodo, kiek elementarių operacijų atlikti, generuojant naujus pirmumo sąrašus.

`Kryzminimas(kiekd:integer; var x,y:intvekt)` – dviejų pirmumo sąrašų (chromosomų) X ir Y vientaškis kryžminimas, kai užduočių skaičius KIEKD.

`Mutacija(var Kiek:integer; var x,y:intvekt)` – Mutacijos procedūra pirmumo sąraše, kai genas matuoja su tikimybe P, t.y. iš pirmumo sąrašo X su KIEK užduočių gaunamas naujas sąrašas Y, sukeičiant atrinktas mutavimui užduotis su atsitiktinėmis užduotimis.

`popkryzminimas(kiekd,kiekchr:integer; var x: popmas)` – Visos populiacijos kryžminimo procedūra, kai iš KIEKCHR chromosomų populiacijos X su KIEKD užduočių kryžminant chromosomas poromis gaunama nauja populiacija. Kreipiamasi į procedūrą **`Kryžminimas.`**

`popmutacija(kiekd,kiekchr:integer; var x: popmas; var kiekmut : integer)` – Visos populiacijos mutavimo procedūra, kai KIEKCHR chromosomų populiacijoje X su KIEKD užduočių

chromosomos mutavimui atrenkamos su tikimybe p_mutac. Gaunamas ir mutavusių chromosomų skaičius KIEKMUT. Kreipiamasi į procedūras **`PirmSarKeit`** ir **`Leistinas`** (keičiamas pirmumo sąrašas kol gaunamas leistinas).

`popmutacija2(kiekd,kiekchr:integer; var x: popmas; var kiekmut : integer)` – Visos populiacijos mutavimo procedūra, kai KIEKCHR chromosomų populiacijoje X su KIEKD užduočių genai iš chromosomos mutavimui atrenkami su tikimybe p_mutac. Gaunamas ir mutavusių genų skaičius KIEKMUT. Kreipiamasi į procedūras **`PirmSarKeit2`** ir **`Leistinas`** (keičiamas pirmumo sąrašas, perkeliant ar sukeičiant užduotis, t.y. genus, kol gaunamas leistinas).

E.2 Programos su realizuotu MA algoritmu su kintama aplinka veikimo schema

Nuskaitoma, kiek uždavinių yra tam tikroje klasėje (480 arba 600).

Kartojama visai uždavinių klasei:
(480 ar 600 kartų, priklausomai nuo uždavinių klasės)

{{
Nuskaitomi visi pradiniai uždavinio duomenys – žinomas optimumas ar žinoma geriausioji tikslo funkcijos reikšmė (to reikia efektyvumo įverčiams nustatyti),
iteracijų skaičius, įtakojantis maksimalų peržiūrėtų pirmumo sąrašų skaičių (sąrašų skaičius 1000 arba 5000), užduočių skaičius, resursų rūšių skaičius, turimų resursų ribos, užduočių trukmės, užduotims atlikti reikalingi resursai, informacija apie nuoseklumo sąryšius, užpildant nuoseklumo sąryšių matricas.

Fiksuojamas vieno uždavinio sprendinio optimizavimo **pradžios laikas**.

Suskaičiuojami visų užduočių kritiniai laikai (priešsrovio algoritmu) bei randamas projekto kritinis laikas.
Sudaromas kritinių kelių pirmumo sąrašas.
NDekoder dekoderiu sudaromas tvarkaraštis su kritinių kelių pirmumo sąrašu.

Atsitiktinių sąrašų generavimas palyginimui: Sugeneruojama ne mažiau kaip KART atsitiktinių leistinų pirmumo sąrašų ir iš jų išrenkamas geriausias (Su mažiausia tikslo funkcijos reikšme)
Nuskaitoma pradinė temperatūra, pradinis aplinkos gylis, priklausantis nuo užduočių skaičiaus ir numatomų iteracijų skaičiaus.

MODELIUOJAMOJO ATKAITINIMO algoritmo realizavimas:
{
Naudojant procedūrą **gylio_sk**, įvertinant kintamą aplinką, suskaičiuojamas elementarių operacijų skaičius RO, kuris atskirais atvejais yra pastovus, t.y. RO=1, RO=10, o kintamos aplinkos atveju RO kinta naudojant atnaujinimo funkciją.

Atliekame RO elementarių operacijų, generuodami (**PirmSarKeit**) naujus pirmumo sąrašus tol, kol bus gauta RO leistinų pirmumo sąrašų (naudodami procedūrą Leistinas tikriname sąrašų leistinumą).

Naudojant dekodavimo procedūrą **NDekoder**, gaunama tikslo funkcijos reikšmė.

Naudojamas Metropolio patvirtinimo kriterijus (geresnė tikslo funkcijos reikšmė patvirtinama, o blogesnė gali būti patvirtinta su tam tikra tikimybe, kuri priklauso nuo temperatūros).
Įsimenama geriausioji pasiekta tikslo funkcijos reikšmė, jei proceso eigoje būtų patvirtintas pirmumo sąrašas su blogesne tikslo funkcijos reikšme.
Atnaujinamos temperatūros *T* ir aplinkos gylio RO reikšmės
}

Fiksuojamas optimizavimo pabaigos laikas, skaičiuojamas skirtumas tarp geriausios žinomos tikslo funkcijos reikšmės ar optimumo ir geriausios reikšmės populiacijoje, fiksuojamas iteracijų skaičius, kurių prireikė reikšmei pasiekti.

}}

Suskaičiuojami bendri algoritmo efektyvumo rodikliai visai uždavinių klasei – kiek procentų atvejų buvo surastas optimumas ar geriausia žinoma tikslo funkcijos reikšmė, kiek vidutiniškai pirmumo sąrašų reikėjo peržiūrėti, kiek laiko vidutiniškai truko optimizacija, koks bendras (išsprendus visą uždavinių klasę) procentinis nuokrypis nuo optimumo ar geriausios žinomos tikslo funkcijos reikšmės ar nuo kritinio kelio.

Visų optimizavimo rezultatų įrašymas į failą.

E.3 Programos su realizuotu GA algoritmu veikimo schema

Nuskaitoma, kiek uždavinių yra tam tikroje klasėje (480 arba 600).

Kartojama visai uždavinių klasei:
(480 ar 600 kartų, priklausomai nuo uždavinių klasės)

{{
Nuskaitomi visi pradiniai uždavinio duomenys – žinomas optimumas ar žinoma geriausioji tikslo funkcijos reikšmė (to reikia efektyvumo įverčiams nustatyti),
Mutacijs tikimybė, chromosomų populiacijoje skaičius, iteracijų skaičius, įtakojantis maksimalų peržiūrėtų pirmumo sąrašų (chromosomų) skaičių (sąrašų skaičius 1000 arba 5000), užduočių skaičius, resursų rūšių skaičius, turimų resursų ribos, užduočių trukmės, užduotims atlikti reikalingi resursai, informacija apie nuoseklumo sąryšius, užpildant nuoseklumo sąryšių matricas.

Fiksuojamas vieno uždavinio sprendinio optimizavimo **pradžios laikas**.

Sugeneruojama pirmumo sąrašų populiacija, naudojant procedūra **PirmSarGen**.

Evoliucinio proceso realizacija vienam uždaviniui – kartojama ne daugiau, nei duotas iteracijų skaičius arba kol pasiekiama geriausia žinoma tikslo funkcijos reikšmė ar optimumas:
{
Atliekamas populiacijos **kryžminimas**, naudojant procedūrą **popkryzminimas**.
Atliekama populiacijos **mutacija**, naudojant procedūrą **popmutacija2**.
„Tėvų" ir „Vaikų" populiacijų **apjungimas** atrankai.
Atranka vykdoma dvikovų metodu, įvertinant tikslo funkcijos reikšmę, gautą procedūra **NDekoder**.
Įsimenama geriausia tikslo funkcijos reikšmė populiacijoje ir ją atitinkantis pirmumo sąrašas.
}

Fiksuojamas optimizavimo pabaigos laikas, skaičiuojamas skirtumas tarp geriausios žinomos tikslo funkcijos reikšmės ar optimumo ir geriausios reikšmės populiacijoje, fiksuojamas iteracijų skaičius, kurių prireikė reikšmei pasiekti.

}}

Suskaičiuojami bendri algoritmo efektyvumo rodikliai visai uždavinių klasei – kiek procentų atvejų buvo surastas optimumas ar geriausia žinoma tikslo funkcijos reikšmė, kiek vidutiniškai pirmumo sąrašų reikėjo peržiūrėti, kiek laiko vidutiniškai truko optimizacija, koks bendras (išsprendus visą uždavinių klasę) procentinis nuokrypis nuo optimumo ar geriausios žinomos tikslo funkcijos reikšmės ar nuo kritinio kelio.

Visų optimizavimo rezultatų įrašymas į failą.