KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

Julius Simonavičius

GAMYBINIŲ TVARKARAŠČIŲ SUDARYMO UŽDAVINIO ALGORITMAI IR ANALIZĖ

Magistro darbas

Vadovas
doc. dr. N. Listopadskis

KAUNAS, 2007

KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

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

GAMYBINIŲ TVARKARAŠČIŲ SUDARYMO UŽDAVINIO ALGORITMAI IR ANALIZĖ

Taikomosios matematikos magistro baigiamasis darbas

Vadovas
( ) doc. dr. N. Listopadskis
2007 06 03

Recenzentas
( ) doc. dr. K. Plukas
2007 06 01

Atliko
FMMM-5 gr. stud.
( ) J. Simonavičius
2007 05 25

KAUNAS, 2007

KVALIFIKACINĖ KOMISIJA

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

Sekretorius – Eimutis Valakevičius, docentas.

Nariai:

- Algimantas Jonas Aksomaitis, profesorius,
- Arūnas Barauskas, dr.,UAB „Elsis“ generalinio direktoriaus pavaduotojas,
- Vytautas Janilionis, docentas,
- Zenonas Navickas, profesorius,
- Vidmantas Povilas Pekarskas, profesorius,
- Rimantas Rudzkis, habil.dr., banko „NORD/LB“ vyriausiasis analitikas.

Simonavičius J. Gamybinių tvarkaraščių sudarymo uždavinio algoritmai ir analizė: taikomosios matematikos magistro darbas / darbo vadovas doc. dr. N. Listopadskis, taikomosios matematikos katedra, fundamentaliųjų mokslų fakultetas, Kauno technologijos universitetas. Kaunas, 2007. 87p.

SANTRAUKA

Darbo pradžioje supažindinsiu su gamybinių tvarkaraščių sudarymo uždaviniu. Jo tikslas rasti tvarkaraštį tam tikrai gamybinei situacijai. Uždavinys turi pilnai nusakyti kas ir kur turi įvykti ir apibrėžti visus apribojimus, o gautas sprendinys turi tenkinti šiuos reikalavimus ir vienareikšmiškai nusakyti operacijų vykdymą. Ši problema aktuali gamyklose, personalo valdyme, krovinių gabenime, oro uostuose, traukinių stotyse ir daugelyje kitų. Kadangi matematinis gamybinių tvarkaraščių sudarymo apibūdinimas sudaromas atsižvelgiant į realaus pasaulio problemas, egzistuoja daug šio uždavinio variantų. Dėl to teko pasirinkti kurį konkretų uždavinį nagrinėti. Pirmajame skyriuje supažindinu su šiuo konkrečiu uždaviniu, pateikiu apibrėžimus, sąvokas ir egzistuojančias problemas sprendžiant gamybinių tvarkaraščių uždavinį. Galiausiai parodysiu, kad tai yra sunkiai sprendžiamas ir dėl to vienas iš aktualių kombinatorinio optimizavimo uždavinių.

Tuomet plačiau apibūdinu genetinį ir skruzdžių kolonijos optimizavimo algoritmus. Šie algoritmai ir naudojami sprendžiant mano konkretų gamybinių tvarkaraščių sudarymo uždavinį. Aš apibūdinu visus parametrus ir koeficientus. Vėliau aš pristatau sukurtą programinę įrangą, skirtą rasti spręsti gamybinių tvarkaraščių sudarymo uždavinį ir vaizdžiai pateikti gautus sprendinius. Taip pat grafikų lentelių ir kitų priemonių pagalba pateikiu atliktų eksperimentų rezultatus. Tuomet apžvelgiu gautus rezultatus ir aptariu pastebėtas tendencijas ir algoritmų veikimo specifiką.

Galiausiai darbo pabaigoje gauti rezultatai bus apibendrinti ir pateiktos gautos išvados apie algoritmų realizaciją ir jų specifiką sprendžiant gamybinių tvarkaraščių sudarymo uždavinį.

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

1. „Gamybinių tvatkaraščių genetinio algoritmo tyrimas“, Matematika ir matematikos dėstymas, 2006;
2. „ Gamybinių tvatkaraščių sudarymo uždavinys“,VI studentų konferencija KTU, 2006;
3. “Tvarkaraščio sudarymo algoritmo analizė”, Lietuvos matematikų draugijos XLVII konferencija, 2006.

Taip pat pateiktas straipsnis „Gamybinių tvarkaraščių sudarymo uždavinio algoritmų palyginimas“ Lietuvos matematikų draugijos XLVIII konferencijai.

Simonavičius J. Algorithms and analysis of the scheduling problem: Masters's work in applied mathematics / supervisor dr. assoc. prof. N. Listopadskis; Department of Applied mathematics, Faculty of Fundamental Sciences, Kaunas University of Technology. – Kaunas, 2007. – 87 p.

SUMMARY

At the begining of this work introduction to the scheduling problem and its basics is given. The goal of a scheduling problem is to make a schedule for a certain production situation. In the problem it is stated what must take place, and the solution describes exactly what should happen at what time. These problems occur in factories, personnel planning, transportation, airfields, railroad stations etcetera. Since mathematical descriptions of scheduling problems are often distilled from practical situations there are many variants of scheduling problems. A selection had to be made which problem is going to be a target of the study. The job shop scheduling problem was chosen. In the first chapter there is definitions of the problems we are trying to solve, introduction of important concepts (properties, bounds, definitions) from the field of scheduling. The last section takes a small detour into theoretical computer science in order to make precise that scheduling problems are hard to solve.

In the second chapter introduction of genetic and ant colony optomization algorithms and its basius is given. It is used to solve scheduling problem, which was mentioned before. Introduction of all genetic and ant colony optimization algorithm operators and settings are given here. Then follows the introduction to software witch was made to solve and visualize solutions of scheduling problem. A great number of plots and figures are used in the experimental chapter to explain what experiments has been done and what the results are.

As usual everything said will be rephrased and put together in the conclusion chapter. Findings for the problem are summarized in this section. Conclusions are made about realization of algorithms and its specifics in solving scheduling problem.

Based on the problems and algorithms of this work, these publications were presented:

1. „Study of genetic algorithm for machine scheduling problem“, Mathematics and its lecturing, 2006;
2. „ Scheduling problem“,VI student conference KTU, 2006;
3. “The Analysis of machine scheduling problem algorithm”, XLVII conference of Lithuanian Mathematicians Association, 2006.

Also a publication „Comparison of algorithms for the scheduling problem“ was submited to the XLVIII conference of Lithuanian Mathematicians Association.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

1. ĮVADAS

Pramonėje, komercinėje veikloje ir daugelyje kitų veiklos sričių dažnai keliami uždaviniai, susiję su veiklos efektyvumu. Pavyzdžiui, architektas nori žinoti kaip suprojektuoti gamyklos patalpas, kad atstumai tarp įrenginių būtų kuo mažesni, laivų planuotojas nori suprojektuoti laivo patalpas taip, kad jos būtų kuo efektyviau panaudojamos gabenant krovinius. Telekomunikacijų specialistai nori žinoti kaip siusti signalą kanalais, kad klaidos tikimybė būtų kuo mažesnė. Visuotinio transporto strategai nori žinoti kaip sudaryti autobusų, traukinių tvarkaraščius, kad keleivių pervežimas būtų kuo efektyvesnis. Taip pat gamyklose labai svarbu paskirstyti operacijas laike taip, kad bendras gamybos laikas būtų kuo mažesnis, nes papildomas gamybos laikas reiškia bereikalingas papildomas sąnaudas. Šviesoforų signalų sinchronizacija, taip pat įvairiu kitų srautų valdymas ir daugybė kitokių gyvenimiškų uždavinių [7].

Tokius uždavinius spręsti tampa ypač sudėtinga, kai didėja jo apimtis arba struktūra. Sprendžiant įprastais metodais, dažnai neįmanoma pateikti net apytikslio atsakymo. Tai ir yra sritis kurioje kombinatorinis optimizavimas praverčia labiausiai.

Kombinatorinis optimizavimas – tai įvairių modelių ir metodų taikymas diskrečios struktūros uždaviniams. Vienas iš svarbiausių veiksnių davusių svarų postūmį sprendžiant kombinatorinio optimizavimo uždavinius – spartus kompiuterinių technologijų vystimasis pastaruosius keletą dešimtmečių. Kadangi kompiuteriams būtina uždavinį suformuluoti diskrečiai, kombinatorinis optimizavimas tapo neatskiriama informatikos dalimi [7].

Savo darbe nagrinėsiu konkretų kombinatorinio optimizavimo uždavinį – gamybinių tvarkaraščių sudarymą. Šio uždavinio sprendimui naudosiu, analizuosiu ir tarpusavyje lyginsiu genetinį ir skruzdžių kolonijos optimizavimo algoritmą ir jų modifikacijas.

1.1 GAMYBOS TVARKARAŠČIŲ SUDARYMO UŽDAVINYS

Gamybos tvarkaraščių sudarymo uždavinys (toliau bus vadinamas tvarkaraščių sudarymu, *angl. schedule - tvarkaraštis*) – paskirstyti ribotus resursus užduotims diskrečioje sistemoje. Toliau kalbėsime apie gamybos tvarkaraščių sudarymą. Gamybos tvarkaraščių sudarymo atveju mašina suprantama kaip tam tikra gamybos linijos dalis. Tvarkaraščių sudaryme dažniausiai susiduriama su dviem apribojimais[4]:

a) mašinų pajėgumų ribos,
b) technologiniai eiliškumo apribojimai.

Bet koks rezultatas tenkinantis šias dvi sąlygas galėtų būti vadinamas tvarkaraščių sudarymo uždavinio sprendiniu. Rastas sprendinys turėtų atsakyti į du klausimus:

1. Kokie resursai bus panaudoti įvykdyti konkrečią užduotį?
2. Kada kiekviena užduotis bus įvykdyta?

Kai kuriais atvejais tvarkaraščių sudarymas atsako tik į pirmą arba tik į antrą klausimą.

Reikia pažymėti, kad toks uždavinio sprendimas iš esmės yra daugiau teorinis, nes neįtraukiami jokie dinaminiai gamybos veiksniai. Taigi kokius veiksnius reikėtų nagrinėti? Šie veiksniai yra tokie [4]:

a) turi būti apibrėžtas tam tikrame konkrečiame laike ir turi būti pajėgus prisitaikyti,
b) jis turi nustatyti vienu metu apdorojamo paketo dydį, turi įvertinti darbo laiko trukmę,
c) turi būti labai lankstus skirtingų gamybos sąlygų atžvilgiu,
d) jis turi atstovauti stochastinę aplinką, kuriai sudaromi tvarkaraščiai,
e) turi egzistuoti kriterijai, kurių pagalba galima būtų tvarkaraščius tarpusavyje lyginti.

Egzistuoja daug veiksnių, turinčių įtakos tvarkaraščių sudarymui. Galutinis rezultatas priklauso nuo mašinų užimtumo, jų pajėgumo ir kt. Iš esmes pagrindiniai veiksniai, įtakojantys tvarkaraščio generavimą, yra pradiniai apribojimai, t. y. mašinų pajėgumo ir eiliškumo apribojimai.

1.2 PRADINIŲ DUOMENŲ IR APRIBOJIMŲ APIBRĖŽIMAS

Tvarkaraščių sudarymas nagrinėja uždavinius, susijusius su resursų paskirstymu laike, norint atlikti tam tikrą rinkinį užduočių. Kiekvienas tvarkaraščių sudarymo uždavinys yra apibūdinamas trimis komponentėmis [1]:

- Mašinų skaičius *M*, darbų skaičius *J* ir atlikimo laikas kiekvienam darbui kiekvienoje mašinoje matricoje *A* (kitaip tariant, operacijų atlikimo laikai)
- Apribojimų rinkinys *C*, kurį privalo tenkinti kiekvienas tvarkaraštis.
- Funkcija *f*, kuri turi būti minimizuota.

Tvarkaraščių sudarymo uždaviniai yra įdomūs kartu ir iš praktines, ir iš teorinės pusės. Kadangi pasirinkau evoliucinį algoritmą, reikia pasirinkti tokį uždavinį, kuris būtų sąlyginai lengvai apibūdinamas. Reikia suformuluoti labai abstraktų modelį, atitinkantį realaus gyvenimo situaciją.

Paprastumo dėlei tarkime, kad kiekvienas darbas gali turėti tik vieną operaciją su bet kuria mašina. Apdirbimo laikai bus užduodami kaip dydžio *J*M* matrica *A*, kur $A_{j,m}$ yra apdirbimo laikas

darbui *j* su mašina *m*. Visi apdirbimo laikai yra griežtai sveikieji teigiami skaičiai. Tvarkaraščių sudarymo uždavinio tikslas – surasti tvarkaraštį. Toks tvarkaraštis yra pilnai ir vienareikšmiškai apibrėžiamas matricos *S* pagalba, kurios dydis *J***M*, kur $S_{j,m}$ yra operacijos atlikimo pradžios laikas darbui *j* ir *m* mašinai. Tinkamame tvarkaraštyje darbai arba mašinos negali būti naudojami dvejose vietose tuo pačiu metu, todėl būtinai turi būti tenkinami šie apribojimai [1]:

$$\forall i,m\, \mathrm{S_{i,m}} \geq 0 \qquad \textbf{(1.1)}$$

$$\forall i,j,m\ \ (\mathrm{i} \neq \mathrm{j} \wedge \mathrm{S_{i,m}} \leq \mathrm{S_{j,m}}) \Rightarrow \mathrm{S_{i,m}} + A_{i,m} \leq S_{j,m} \qquad \textbf{(1.2)}$$

$$\forall i,m,n\ \ (\mathrm{m} \neq \mathrm{n} \wedge \mathrm{S_{i,m}} \leq \mathrm{S_{i,n}}) \Rightarrow \mathrm{S_{i,m}} + A_{i,m} \leq S_{i,n} \qquad \textbf{(1.3)}$$

Funkcija *f* arba laikas, kai baigiama vykdyti paskutinė operacija, turi būti minimizuota. Jos formulė:

$$\mathrm{f(S)} = \max_{\mathrm{i,m}} (S_{i,m} + A_{i,m}) \qquad \textbf{(1.4)}$$

Ši funkcija dar vadinama gamybos trukmės arba laiko kriterijumi. Žinoma, kartais naudojamos ir alternatyvios funkcijos, tačiau dažniausiai naudojama ši.

Situacija, kurią parodėme viršuje, nusako statinį tvarkaraščių sudarymą. *Statiniame tvarkaraščių sudaryme* visi darbai yra prieinami iš pat pradžių. *Dinaminio tvarkaraščių sudarymo* uždavinyje kiekvienas darbas tampa prieinamas tam tikru laiko momentu, šiuos laikus nusako dimensijos *J* vektorius *r*. Vietoje reikalavimo, kad visos operacijos turi prasidėti po nulinio momento, šis apribojimas turi būti tenkinamas [1]:

$$\forall i,m\, \mathrm{S_{i,m}} \geq r_i$$

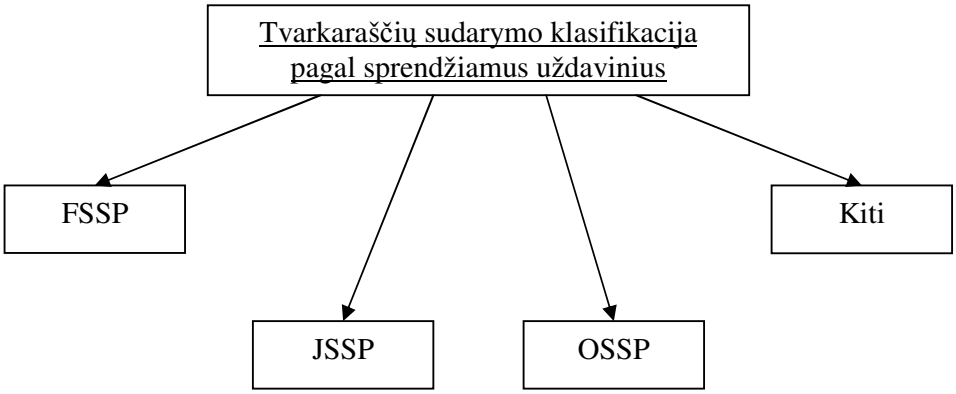
Dauguma metodų, sukurtų tam tikrai konkrečiai tvarkaraščių sudarymo rūšiai, taip pat gali būti pritaikyti kitoms rūšims. Pagrindinis skirtumas yra tame, kad gamybos trukmė nėra labai tinkamas vertinimo kriterijus dinaminio tvarkaraščių sudarymo atveju, nes tuomet suteikiama per daug reikšmės paskutiniam prieinamam darbui.

Kitas ribojimas yra toks, kad ši situacija apibūdina deterministinį tvarkaraščių sudarymą, nes viskas apie uždavinį yra žinoma iš anksto ir nieko nenumatyto negali įvykti. Praktikoje mašinos genda ir siuntos gali būti pavėluotos. Stochastiniame tvarkaraščių sudaryme tokie dalykai taip pat gali būti įvykdyti. Tokie tvarkaraščiai, kurie yra ganėtinai patikimi ir tada, kai įvyksta nenumatyti dalykai, vadinami lanksčiais. Šio tipo tvarkaraščiai yra naudingesni praktikoje, tačiau yra prasčiau pritaikyti palyginimo reikmėms.

2. TEORINĖ DALIS

PAGRINDINIAI TVARKARAŠČIŲ SUDARYMO UŽDAVINIAI

Sekančiuose skyreliuose aptarsiu pagrindinius gamybos tvarkaraščių sudarymo uždavinius. Nors tvarkaraščių tipų yra gerokai daugiau, tačiau nagrinėjant gamybos tvarkaraščių uždavinius, dažniausiai naudojami būtent šie.



2.1 pav. Tvarkaraščių sudarymo klasifikacija pagal sprendžiamus uždavinius

FSSP TVARKARAŠČIŲ SUDARYMAS

FSSP (*angl. flow shop scheduling problem*) atveju visos mašinos gamykloje yra sustatytos vienoje linijoje. Tai reiškia, kad darbai pirmajai mašinai gali būti surikiuoti be apribojimų, bet vėliau jie turi būti apdoroti kiekvienos mašinos ta pačia seka, kuria jie buvo apdoroti pirmos mašinos. Tai gali būti formalizuota tokiu būdu [1]:

$$S \sim C \Leftrightarrow \forall i, j, m, n \ (\mathrm{S_{i,m}} \le S_{j,m} \Rightarrow \mathrm{S_{i,n}} \le S_{j,n}) \qquad \textbf{(2.1)}$$

JSSP TVARKARAŠČIŲ SUDARYMAS

JSSP (*angl. job shop scheduling problem*) tvarkaraščių sudarymo atveju, kiekvienas darbas turi pakliūti į mašinas tam tikra iš anksto numatyta tvarka. Uždavinio apibūdinime turi būti duota dydžio *J*M* matrica *B*. Kiekviena *B* eilutė *j* nusako, kokia tvarka darbas j turi aplankyti mašinas. Tai gali būti formalizuota tokiu būdu [1]:

$$S \sim C \Leftrightarrow \forall i, x, y \ (\mathrm{x} < y \Rightarrow \mathrm{S_{i,B_{i,x}}} \le S_{i,B_{i,y}}) \qquad \textbf{(2.2)}$$

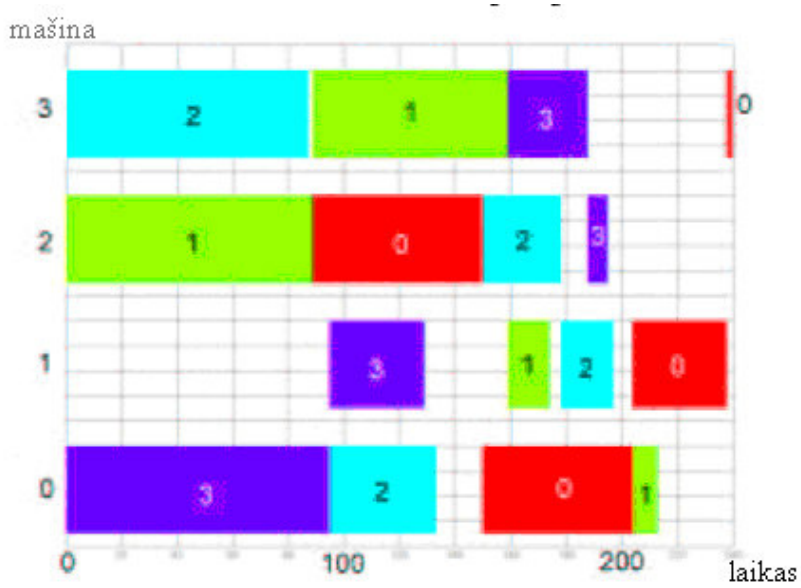
2.1.2.1 JSSP PAVYZDYS

Šiame JSSP pavyzdyje uždavinys apibrėžiamas dvejomis matricomis *A* (apibrėžia gamybos laikus) ir *B* (mašinų aplankymo seka kiekvienam darbui) kurios pateikiamos žemiau. Tai yra uždavinys su keturiais darbais ir keturiomis mašinomis.

$$A = \begin{pmatrix} 54 & 34 & 61 & 2 \\ 9 & 15 & 89 & 70 \\ 38 & 19 & 28 & 87 \\ 95 & 34 & 7 & 29 \end{pmatrix}$$

$$B = \begin{pmatrix} 2 & 0 & 1 & 3 \\ 2 & 3 & 1 & 0 \\ 3 & 0 & 2 & 1 \\ 0 & 1 & 3 & 2 \end{pmatrix}$$

Šio uždavinio sprendinys pateiktas 2.2 paveiksle. Tokia iliustracija yra vadinama *Ganto diagrama*. Kiekviena eilutė atstovauja vieną mašiną. Skirtingų spalvų stačiakampiai atstovauja skirtingus darbus. X ašis reprezentuoja laiko tėkmę. Pavyzdinio tvarkaraščio trukmė 240 laiko vienetų [1].



2.2 pav. Pavyzdinio uždavinio sprendinys

Ganto diagramos gali būti naudojamos tikrinant tvarkaraštį. Pagal ją galima patikrinti ar tvarkaraštis tenkina apribojimus.

OSSP TVARKARAŠČIŲ SUDARYMAS

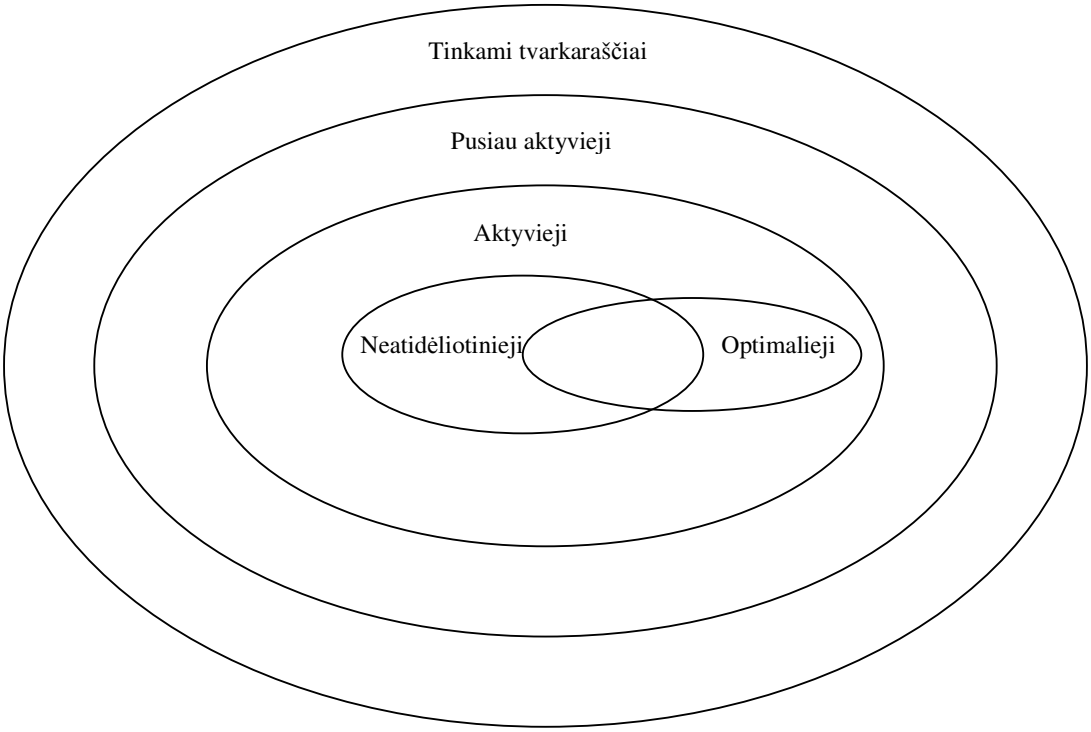
OSSP (*angl. open shop scheduling problem*) tvarkaraščių sudarymo atveju jokių papildomų apribojimų netaikoma, t. y.[1]:

$$C = \phi \qquad \textbf{(2.3)}$$

Bet kuris JSSP ir FSSP tvarkaraštis yra taip pat ir OSSP tvarkaraštis. Atkreipkite dėmesį, kad darbai ir mašinos gali būti laisvai sukeičiami vietomis tam pačiam uždaviniui spręsti. Taigi apie OSSP uždavinį galime galvoti kaip į tvarkaraščio šokių konkursui sudarymą, kur *J* yra berniukų skaičius ir *M* – mergaičių skaičius ir duoti laikai, kuriuos poros turi praleisti kartu.

TVARKARAŠČIŲ TIPAI

Paieškos sritis, kurioje galima nagrinėti tvarkaraščius, yra labai didelė. Dėl to buvo sugalvota keletas apribojimų, kad gero tvarkaraščio radimas būtų paprastesnis. Sekančiame paveiksle pateikiu ryšį tarp skirtingų tvarkaraščių tipų.



2.3 pav. Tvarkaraščių tipų Veno diagrama

Bet koks tvarkaraštis, tenkinantis apribojimus, įvestus tam tikro tiriamojo uždavinio, vadinamas *tinkamu sprendiniu arba tvarkaraščiu* (*angl. feasible schedules*). Bet koks tinkamas tvarkaraštis gali

būti perdarytas į *pusiau aktyvųjį* jei visos darbų operacijos būtų pradėtos kiek įmanoma anksčiau, nekeičiant jų vykdymo tvarkos. Taigi tvarkaraštis S bus pusiau aktyvusis, jei visiems i, m tvarkaraštis apibrėžtas tokiu būdu [1,2]:

$$T_{i,m} = S_{i,m} - 1 \quad \textbf{(2.4)}$$

$$T_{j,n} = S_{j,n} \text{ jei } (j,n) \neq (i,m) \quad \textbf{(2.5)}$$

nėra tinkamas sprendinys. Jei tvarkaraštis yra pusiau aktyvusis, kiekvieno darbo pradžia turi būti suvaržyta tam tikro apribojimo. Šie apribojimai, kaip jau aptariau anksčiau priklauso nuo to, koks yra nagrinėjamas uždavinys. Pusiau aktyviajame tvarkaraštyje kiekviena operacija prasideda, kai mašina arba darbas tampa prieinami arba pradiniu laiko momentu. Žemiau pateiktos trys sąlygos tą ir apibrėžia. Kiekvienai operacijai (*j,m*) pusiau aktyviajame tvarkaraštyje bent viena iš šių trijų sąlygų turi būti tenkinama [1,2]:

$$S_{j,m} = 0 \quad \textbf{(2.6)}$$

$$\exists n \; S_{j,m} = S_{j,n} + A_{j,n} \quad \textbf{(2.7)}$$

$$\exists k \; S_{j,m} = S_{k,m} + A_{k,m} \quad \textbf{(2.8)}$$

Tam tikras tinkamumo matas vadinamas *reguliariuoju*, jei darbų paankstinimas rezultato nepablogina. Darbo trukmės kriterijus turi šią savybę.

Tvarkaraštį vadiname aktyviuoju, jei nei viena operacija negali būti paankstinta taip, kad nekonfliktuotų su kitomis operacijomis. Operacijų atlikimo seka gali būti keičiama. Tvarkaraštis yra aktyvusis, jei su visais *i,m* ir $0 \le t \le S_{i,m}$ tvarkaraštis *T* apibrėžtas tokiu būdu [1,2]:

$$T_{i,m} = t \quad \textbf{(2.9)}$$

$$T_{j,n} = S_{j,n} \text{ jei } (j,n) \neq (i,m) \quad \textbf{(2.10)}$$

nėra tinkamas sprendinys.

Dar griežčiau apibrėžtas yra *neatidėliotinas tvarkaraštis* (*angl. non-delay schedule*). Tokiame tvarkaraštyje, jei mašina *m* nieko nedaro ir darbas *j* joje nėra vykdomas, remiantis apribojimais, sekančiame žingsnyje darbas galėtų būti vykdomas mašinoje *m*, tuomet jis ir turi būti įvykdytas sekančiame žingsnyje. Tegul $B_M\ (S,m,t)$ reiškia, kad mašina *m* yra užimta laiko momentu *t* ir tegul $B_J\ (S,j,t)$ reiškia, kad darbas j yra vykdomas laiko momentu t tvarkaraštyje S. Formaliai [1,2]:

$$B_M\ (S,m,t) \Leftrightarrow \exists j \; S_{j,m} \le t < S_{j,m} + A_{j,m} \quad \textbf{(2.11)}$$

$$B_J\ (S,j,t) \Leftrightarrow \exists m \; S_{j,m} \le t < S_{j,m} + A_{j,m} \quad \textbf{(2.12)}$$

Taip pat reikia apibrėžti *T* tokiu būdu: $S \sim_t T$, tai reiškia, kad *S* yra lygus *T* iki laiko momento t:

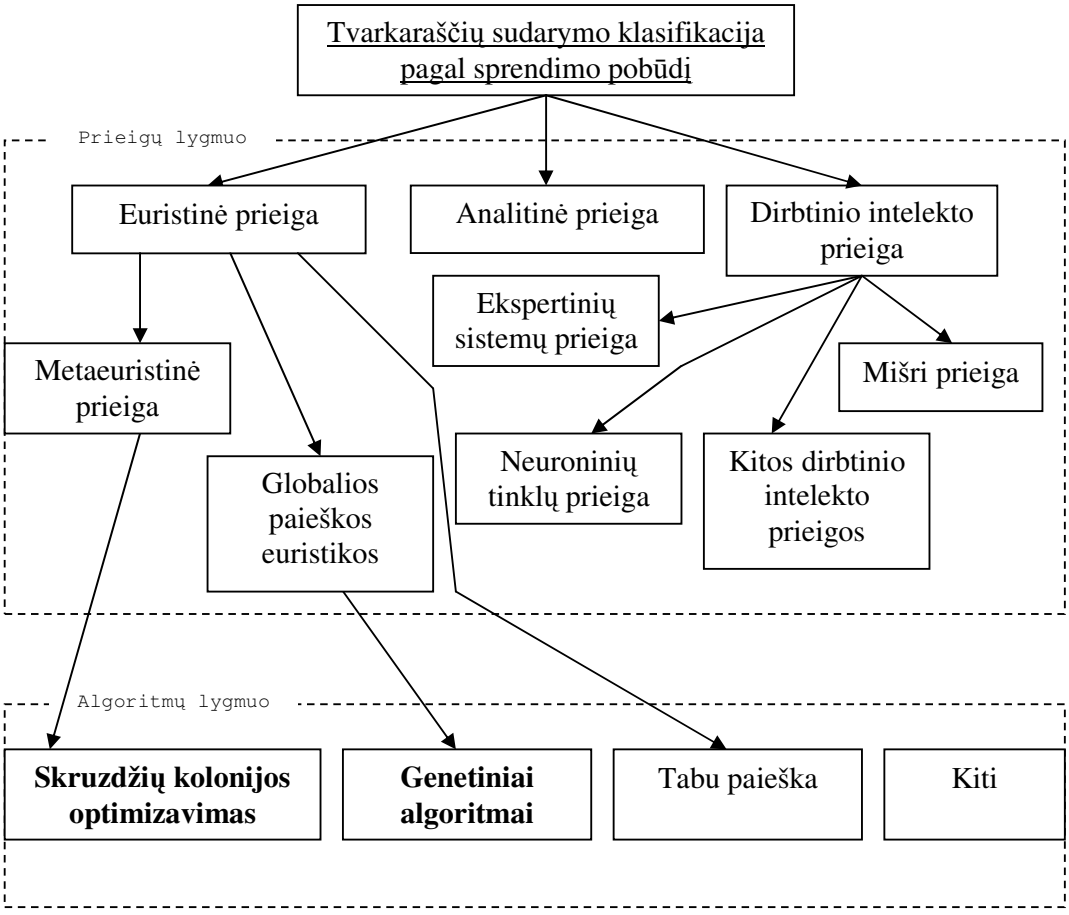
$$S \sim_t T \Leftrightarrow (\forall j,m(S_{j,m} < t \vee T_{j,m} < t) \Rightarrow S_{j,m} = T_{j,m}) \quad \textbf{(2.13)}$$

Dabar apibrėšime tai, kad jei įmanoma pradėti darbą, jis privalo būti pradėtas. *S* yra neatidėliotinas tvarkaraštis, jei tenkinama sąlyga:

$$\forall t, j, m \neg\exists T\ T \sim_t S \wedge T_{j,m} = t \wedge \neg B_M\ (S,m,t) \wedge \neg B_J\ (S,j,t) \quad \textbf{(2.14)}$$

Įmanomas toks variantas, kad visi optimalūs tvarkaraščiai yra ne neatidėliotini. Dažnai JSSP neatidėliotinų tvarkaraščių klasės gamybos laiko vidurkis yra trumpesnis nei aktyvių tvarkaraščių klasės. Kadangi dėl didelių apimčių dažniausiai neįmanoma rasti optimalaus sprendinio, ieškoma aktyvių arba neatidėliotinų tvarkaraščių.

TVARKARAŠČIŲ SUDARYMO UŽDAVINIŲ SPRENDIMO METODAI



2.4 pav. Tvarkaraščių sudarymo klasifikacija pagal uždavinio sprendimo pobūdį

Analitinėje prieigoje naudojamos tokios priemonės kaip: tiesinis programavimas, šakų ir ribų (*angl. branch and bound*) algoritmas, dinaminis programavimas ir kt. Šios priemonės pateikia tikslų atsakymą, tačiau, sprendžiant didelės apimties tvarkaraščių sudarymo uždavinius, jos yra labai neefektyvios. Taip yra netinkami dėl tvarkaraščių problemos sunkumo, apie kurį kalbama 2.4 skyrelyje [6].

Jei uždavinio apimtis yra didelė, verta naudoti euristinius (metaeuristinius) metodus, kai atsakymo bandoma ieškoti remiantis tam tikromis iš anksto numatytomis taisyklėmis. Jei metodas sukonstruotas tinkamai, atsakymas bus ganėtinai tikslus ir naudos gerokai mažiau resursų nei analitiniai metodai [6].

Dirbtinio intelekto prieigą galima suskirstyti į keturis pogrupius: ekspertinių sistemų, neuroninių tinklų, mišri, kitos dirbtinio intelekto prieigos.

Ekspertinė sistema (*arba ES*) dar vadinama žiniomis pagrįsta pagalbos sistema sprendimų priėmimui. ES iš esmės yra programa, kuri bando atkartoti žmogaus eksperto veiksmus tam tikroje siauroje srityje, turėdama žinių duomenų bazę [6].

Plačiai modeliavime naudojami neuroniniai tinklai, taip pat gali būti efektyviai naudojami ir sudarant tvarkaraščius.

Vieni iš efektyviausių metodų sunkiems uždaviniams spęsti (tokiems kaip dauguma tvarkaraščių sudarymo uždavinių) - tai tabu paieškos bei genetiniai algoritmai. Dėl efektyvumo ir lankstumo paieškos uždaviniuose savo darbe tvarkaraščių sudarymo uždaviniui spręsti pasirinkau modifikuotą genetinį algoritmą, apie kurį bus kalbama 3 skyriuje.

UŽDAVINIO SUDĖTINGUMAS

Tvarkaraščių sudarymo uždaviniai yra tinkami spręsti evoliucinių metodų pagalba, nes priimta, kad jie yra sunkiai išsprendžiami. Kompiuterių moksle buvo daug stengiamasi, kad būtų galima tiksliai apibrėžti tai, kas yra sunkus uždavinys. Šiame skyrelyje supažindinsiu su pagrindiniais teiginiais ir terminais, kurių pagalba parodysime, kad OSSP ir JSSP iš tiesų yra sunkiai sprendžiami uždaviniai, kuriems reikalingi aproksimacijos metodai.

Kiekvienu atskiru tam tikro uždavinio atveju galime apibrėžti jo sprendinio *apimtį*. Nagrinėjant tvarkaraščių sudarymo uždavinius, matricos *A* elementų skaičius yra apibrėžiamas kaip sprendinio apimtis, kurią pažymėsime *JM* [1].

Kiekvienam algoritmui reikalingas tam tikras skaičiavimo žingsnių skaičius, kad uždavinys būtų išspęstas. Daugumai algoritmų galima apskaičiuoti žingsnių skaičiaus viršutinį rėžį remiantis uždavinio apimtimi reikalingo tam, kad uždavinys būtų išspręstas. Pavyzdžiui, *n* dydžio skaičių sąrašo

L rikiavimui reikėtų mažiau nei *n*log(n)* žingsnių. Apie tokią lengvai sprendžiamų uždavinių klasę dažnai kalbama kaip apie polinominiame laike sprendžiamų uždavinių klasę. Kitaip tariant, turi egzistuoti tokia polinominė funkcija, kad būtų įmanoma aprėžti iš viršaus žingsnių skaičių *t(S)* konkrečiam uždaviniui *S*. Formaliai tai galima aprašyti tokiu būdu [1]:

$$\forall S \exists a, b\, t(S) < |S|^a + b \qquad \textbf{(2.15)}$$

Sudėtingumo teorijoje NP-išbaigtumo uždaviniai dažniausiai yra pasirinkimo, t.y. tokie, kurie formuluojami taip, kad atsakymas į juos gali būti tik taip arba ne. Tvarkaraščių sudarymo uždavinys yra optimizavimo, t. y. reikia optimizuoti tam tikrą funkciją. Kiekvienam optimizavimo uždaviniui galime suformuluoti atitinkamą apsisprendimo uždavinį, į kurį būtų atsakoma tik taip arba ne. Šiam uždaviniui kaip duomenys, būtų įvedami: konkretus optimizavimo uždavinys ir tam tikras sprendinys. Klausimą suformuluotume tokiu būdu: „Ar gali būti gautas geresnis sprendinys nei *x*?“. Konkrečiai tvarkaraščių sudarymo uždaviniams klausimas būtų toks: „Ar yra toks tvarkaraštis, kurio gamybos trukmė yra trumpesnė nei *x*?“ [1].

Tvarkaraščio gamybos trukmės radimas gali būti atliktas polinominiame laike visiems trims prieš tai aprašytiems tvarkaraščių sudarymo uždaviniams.

Apsisprendimo uždavinį vadiname nedeterministiniu polinominiame laike arba kitaip NP, jei konkretaus atvejo (tvarkaraščių sudarymo uždavinys ir konkreti reikšmė *x*) ir siūlomo sprendinio (tvarkaraščio) pora, gali būti palyginta polinominiame laike. Toks uždavinys, kuriame reikia apsispęsti ar galima rasti tvarkaraštį su trumpesniu gamybos laiku, yra NP, nes gamybos trukmės apskaičiavimas yra atliekamas polinominiame laike [1].

Visiškai nežinoma ar NP uždavinys yra išsprendžiamas polinominiame laike. Tačiau dauguma mokslininkų mano, kad visgi tai yra neįmanoma. Konkretūs NP uždavinių atvejai (pvz. tvarkaraščių sudarymo uždavinys) vadinami NP-išbaigtais uždaviniais. Manoma, kad šių uždavinių neįmanoma išspręsti polinominiame laike, nes kitu atveju daug sudėtingų uždavinių taptų lengvai išsprendžiamai [1].

Iš JSSP ir OSSP suformuluoti apsisprendimo uždaviniai, kaip ir daugelis kitų uždavinių, yra NP-išbaigti. Jei apsisprendimo uždavinys yra NP-išbaigtas, tai to paties uždavinio optimizavimas yra NP-sunkus uždavinys. Taigi JSSP ir OSSP yra NP-sunkūs uždaviniai, o tai reiškia, kad jiems spręsti verta naudoti aproksimuojančius algoritmus.

3. TIRIAMOJI DALIS IR REZULTATAI

GENETINIS ALGORITMAS

Šiame darbe tvarkaraščių sudarymo uždaviniui, kurį tiksliai apibrėšiu sekančiame skyrelyje, naudosiu genetinį hibridinį algoritmą, t. y. genetinį algoritmą kartu su vietinės paieškos algoritmu, kuris leidžia patikslinti genetinio algoritmo darbo rezultatus.

Genetiniai algoritmai yra prisitaikantys ir gali būti naudojami sprendžiant paieškos ir optimizavimo uždavinius. Šis algoritmas buvo sugalvotas remiantis genetiniais procesais vykstančiais gamtoje. Per daugelį kartų natūrali populiacija evoliucionuoja remiantis natūralios atrankos principu, t. y. stipriausio individo išlikimo principu, kuris pirmą karta buvo aprašytas Čarlzo Darvino knygoje „Rūšių pradmenys“. Mėgdžiojant šį procesą, genetiniai algoritmai yra pajėgūs prisitaikyti prie realių gyvenimiškų situacijų ir rasti jų sprendinius, jei algoritmas buvo tinkamai sudarytas [5,7].

Prieš pradedant spręsti uždavinį, pirmiausiai reikia rasti tinkamą kodavimą (arba atstovavimą) formuluojamam uždaviniui. Taip pat reikalinga *patikrinimo funkcija*, su kuria galime įvertinti tam tikrų individų (arba tiesiog sprendinių) gerumą, tam kad būtų įmanoma du individus palyginti tarpusavyje. Genetinio algoritmo realizacijos vykdymo metu pastoviai yra atrenkami du tėvai, kurie kryžminami tarpusavyje, kad gauti palikuonį.

Tariame, kad potencialūs uždavinio sprendiniai gali būti atstovaujami parametrų rinkinio. Šie parametrai (kitaip vadinami genais) yra sujungiami kartu ir tokiu būdu gaunama tam tikra reikšmių seka (chromosoma). Genetinėje terminologijoje, šis parametrų rinkinys atstovaujamas vienos chromosomos yra vadinamas *individu*. Individo tinkamumas, kuris yra nustatomas patikrinimo funkcijos pagalba, priklauso nuo jo chromosomos.

Individai vienos iteracijos (kartos) metu yra atrenkami ir kryžminami tarpusavyje ir tokiu būdu gaunami palikuonys, kurie patenka į sekančia populiaciją. Tėvai yra atsitiktinai atrenkami iš populiacijos, pagal schemą, kuri teikia pirmenybę geresniems individams. Išsirinkus du tėvus, jų chromosomos yra apjungiamos naudojantis *kryžminimo* ir *mutacijos* procedūromis. Mutacija taikoma keletui individų tam, kad išlaikyti populiacijos įvairovę. Pateikiu standartinio genetinio algoritmo pseudo-kodą [5,7]:

```
Genetinis Algoritmas
{
   Generuoti pradinę populiaciją Pt
   Patikrinti populiaciją Pt
```

```
    Kol sustojimo kriterijus netenkinamas Kartoti
    {
       Išrinkti elementus iš Pt kurie bus kopijuojami į Pt+1
       Kryžminti elementus iš Pt ir įdėti į Pt+1
       Mutuoti elementus iš Pt ir įdėti į Pt+1
       Patikrinti naują populiaciją Pt+1
       Pt = Pt+1
    }
}
```

ATSTOVAVIMAS CHROMOSOMOMIS IR DEKODAVIMAS

Chromosoma = ($genas_1$, $genas_2$, $genas_3$, ..., $genas_n$)

Mano pasirinktu konkrečiu tvarkaraščių sudarymo atveju, geną apibrėšime kaip vieną sprendinio matricos (tvarkaraščio) stulpelį. Chromosoma (sprendinys) - tai šių genų rinkinys, kuris vienareikšmiškai atstovauja kiekvieną populiacijos individą. Todėl turint chromosomą, patikrinimo funkcijos pagalba galime įvertinti individą.

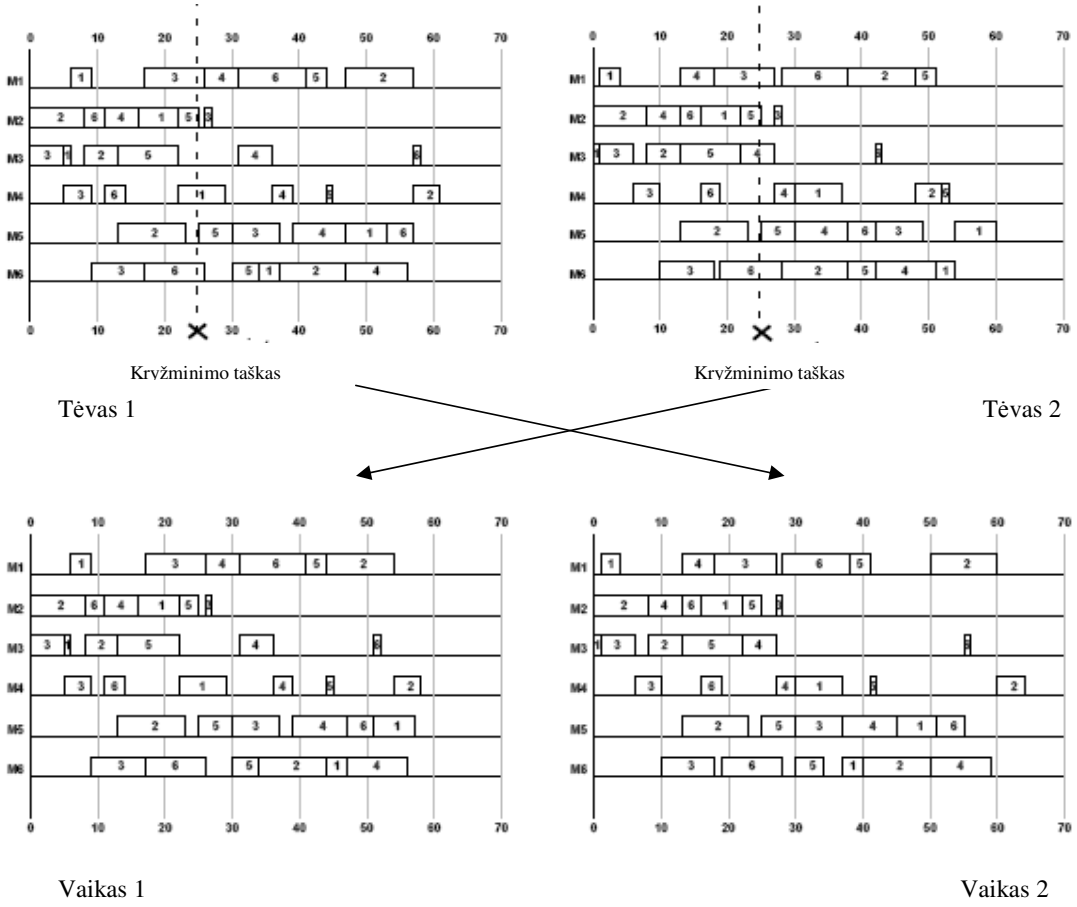
Įgyvendinant algoritmą, taip bus naudojamas atsitiktinės chromosomos generatorius. Taip pat bus atliekami mutacijos, kryžminimo ir kiti veiksmai. Po bet kurio veiksmo su populiacijos individais, jie būtinai turi išlikti tinkamais sprendiniais pagal konkretaus uždavinio formuluotę, priešingu atveju, galime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

EVOLIUCINĖ STRATEGIJA

Kad gauti gerus palikuonis, genetiniu algoritmu yra apdorojama atsitiktinai sugeneruota pradinė populiacija. Gali būti daugybė genetinio algoritmo variantų, keičiant mutacijos, kryžminimo ir kitus reprodukcijos operatorius. Atrinkimo ir kryžminimo operatoriai nusako, kurie individai taps tėvais ir kurie tėvų genai bus naudojami formuojant jų palikuonį. Mutacijos operatorius suteikia atsitiktinius pokyčius populiacijoje. Dažniausiai kryžminimo ir parinkimo operatoriai įtakoja populiacijos kokybę ir konvergavimo greitį. Tačiau mutacijos operatorius atvirkščiai – neleidžia metodui konverguoti per greitai (t.y. jei tinkamai parinktas, neleidžia konverguoti į lokalų minimumą), taip pat įveda naujos informacijos į populiaciją [3].

Reprodukcijos metu pirmiausiai išrenkami keletas pačių geriausių individų kurie pateks ir į sekančią populiaciją (tai vadinama *elitine strategija*). Tokios reprodukcijos privalumas prieš paprastą tikimybinę reprodukciją yra tai, kad geriausias populiacijos individas monotoniškai mažėja kiekvienos iteracijos metu. Tačiau jei parametrai netinkamai parinkti, gali pasireikšti ir elitinės strategijos trūkumas, t. y. konvergavimas į lokalų minimumą. Didesnės mutacijos parametro reikšmės dažniausiai leidžia šio trūkumo išvengti [3].

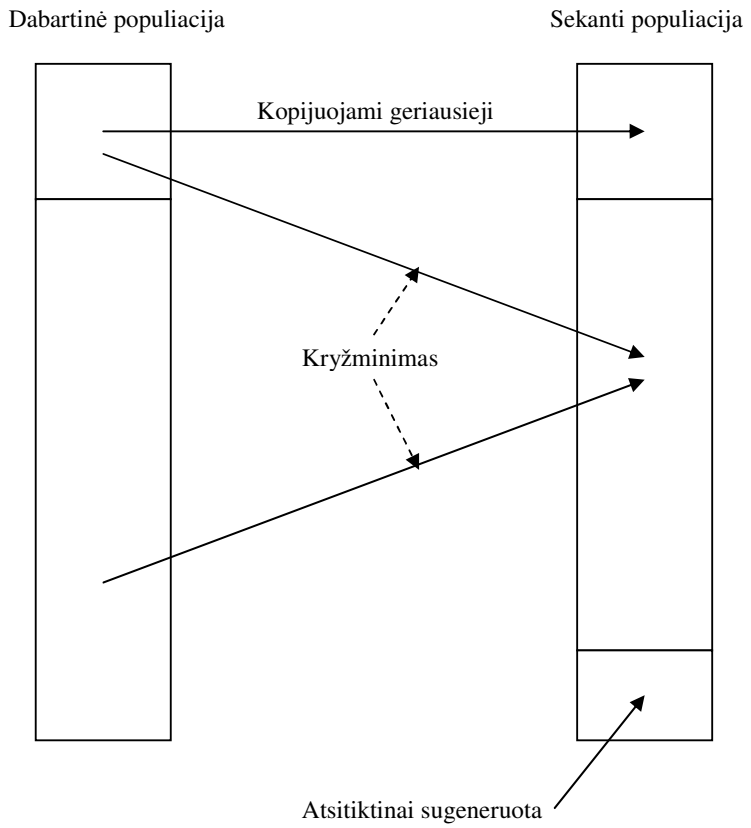
Bus naudojami trijų tipų kryžminimai: vienataškis, daugiataškis ir normaliojo skirstinio. Šie kryžminimo tipai bus naudojami eksperimentų vykdyme ir lyginami tarpusavyje. Sekančiame paveiksle pateikiamas vienataškio kryžminimo pavyzdys. Atsitiktinai parenkamas kryžminimo taškas ir dvi tvarkaraščių dalys apjungiamos į vieną tvarkaraštį vaiką [3].



3.1 pav. Vienataškio kryžminimo pavyzdys

Analogiškai ir daugiataškiame kryžminime, tačiau kryžminimo taškų yra daugiau nei vienas. Kurią tvarkaraščio dalį naudoti iš kurio tėvo nusprendžiama pagal tam tikrą vartotojo įvestą monetos kritimo tikimybę. Kryžminant pagal normalųjį skirstinį, kurią dalį tvarkaraščio paimti iš kurio tvarkaraščio, sprendžiama pagal standartinį normalųjį skirstinį.

Vietoj įprastinės mutacijos, kai gali mutuoti kiekvieno individo kiekvienas genas pagal tam tikrą mažą tikimybę, naudosiu mutaciją, kurios metu iš populiacijos išrenkami patys blogiausi individai ir vietoje jų atsitiktinai sugeneruojami nauji individai, kurie užima senųjų vietą naujojoje populiacijoje. Tokia mutacija padės lengviau išvengti ankstyvo konvergavimo į lokalų minimumą [2].



3.2 pav. Principinė genetinio algoritmo vienos iteracijos schema

SKRUZDŽIŲ KOLONIJOS OPTIMIZAVIMO ALGORITMAS

Pirmoji skruzdžių kolonijos metaeuristika, pavadinta skruzdžių sistema (AS), buvo įkvėpta skruzdžių elgsenos studijų. Skruzdės bendrauja tarpusavyje feromonų pagalba. Feromonai – tai medžiaga, kurios skruzdės vaikščiodamos palieka ant žemės. Buvo nustatyta, kad kuo daugiau skruzdės naudojasi tam tikru keliu, tuo daugiau šis kelias tampa patrauklus kitom skruzdėm maisto ieškojimui, dėl padidėjusio feromonų kiekio. Jei kliuvinys staiga padedamas ant šio kelio, skruzdes iš pradžių eis į kairę arba į dešinę praktiškai atsitiktiniu principu, tačiau tos skruzdės, kurios pasirenka trumpesnį kelią, kelionėje užtruks trumpiau ir šiuo maršrutu per tą patį laiką pereis daugiau skruzdžių,

o tai reiškia, kad feromonų šiame maršrute bus palikta daugiau. Taigi šis kelias taps patrauklesnis ir kitoms skruzdėms [9,12].

Taigi suformuojame *m* dirbtinių skruzdžių, kurios atsitiktiniu principu darydamos sprendimus ieško tinkamų sprendinių. Tai kartoja *n* kartų. Tos skruzdės kurios randa tinkamą sprendinį, savo kelią pažymi tam tikru kiekiu feromonų. Sekančioje iteracijoje skruzdes minėtieji feromonai „traukia“, t.y. tikimybė kad skruzdė pasirinks būtent tokį maršrutą padidėja. Taip pat dažniausiai reikalinga tam tikra papildoma euristika, tam kad, skruzdės teisingai pasirinktų kryptį priklausomai nuo problemos specifikos. AS skruzdės turi atmintį (tabu sąrašą), kuris saugo jau aplankytus komponentus dabartiniame kelyje. Reikia pabrėžti, kad tai nėra tas pats tabu sąrašas, kuris naudojamas tabu paieškos algoritme [9,12].

Sukonstruojame optimizavimo problemos grafinę reprezentaciją *G*. Metaeuristika pradeda darbą suteikdama kiekvienai *G* kraštinei tam tikrą realųjį feromonų kiekį *c*. Tuomet kiekvienai iš *m* skruzdžių dažniausiai atsitiktinai priskiriama starto pozicija, kuri įtraukiama į tabu sąrašą. Tuomet kiekviena skruzdė taikys (3.1) kol galiausiai suras tinkamą sprendinį. Kai kiekvienos skruzdės tabu sąrašas bus pilnas, jos nueitą atstumą įvertinsime funkcijos $\Phi_{\Psi}(t)$ pagalba ir geriausias rezultatas išsaugojamas. Feromonų kiekis pagal kraštinę (*i,j*) perskaičiuojamas remiantis (3.2). Pagaliau visi tabu sąrašai ištuštinami ir jei pabaigos kriterijus netenkinamas, tęsiame toliau [9,12].

Kiekvienos skruzdės sprendimas grindžiamas feromonų kiekiu $\tau_{i,j}$, esančio ant kraštinės (*i,j*), bet taip pat gali būti įtraukiamas ir euristinis atstumas $\eta_{i,j}$ pagal atitinkamą kraštine. Perėjimo tikimybė iš *i* į *j* elementą *k*-tajai skruzdei laiko momentu *t* apibrėžiama [9]:

$$P^{k}{}_{i,j} = \begin{cases} \dfrac{[\tau_{i,j}(t)]^{\alpha}[\eta_{i,j}]^{\beta}}{\sum [\tau_{i,k}(t)]^{\alpha}[\eta_{i,k}]^{\beta}} \; jei (i,j) \notin tabu_k \\ k \in leidžiami_k \\ 0 \end{cases} \qquad \textbf{(3.1)}$$

kur $tabu_k$ yra neleidžiami perėjimai, o $tabu_k = \{C - tabu_k\}$. Parametrai α ir β yra programuotojo ar naudotojo nustatomi parametrai, kurie nustato kokia įtaką turi feromonai ir euristinis atstumas. Nustačius $\beta = 0$, bus naudojama tik feromonų informacija ir atvirkščiai.

Skruzdės feromonus paliks arba „žingsnis po žingsnio“ arba pavėlinto feromonų atnaujinimo principu. Dažna praktika yra atnaujinti feromonus tokiu būdu [9]:

$$\tau_{i,j}(t+n) = \rho \cdot \tau_{i,j} + \Delta\tau_{i,j} \qquad \textbf{(3.2)}$$

kur ρ yra realusis koeficientas, toks kad $(1-\rho)$ reprezentuoja *išgaravimo koeficientą* feromonų pagal kraštinę (*i,j*) tarp laiko momento *t* ir *t+n*. Kad feromonai nesikauptų pergrietai turėtų galioti $\rho \in (0,1)$.

Visi feromonai palikti *m* skruzdžių $\Delta\tau_{i,j}$ yra apskaičiuojami toki būdu [9]:

$$\Delta\tau_{i,j} = \sum_{k=1}^{m} \Delta\tau^{k}{}_{i,j} \quad \textbf{(3.3)}$$

kur $\Delta\tau_{i,j}$, (t.y. feromonų kiekis paliktas *k*-tosios skruzdės) apskaičiuojamas [9]:

$$\Delta\tau^{k}{}_{i,j} = \begin{cases} \dfrac{Q}{L_k} \; jei\; k-toji\; skruzdė\; keliauja\; (i,\, j)\; kraštine \\ 0 \end{cases} \quad \textbf{(3.4)}$$

tarp laiko momentų *t* ir *t+n*, kur Q yra teigiama konstanta ir L_k yra *k*-tosios skruzdės kelio ilgis. Q/L duoda feromonų kiekį vienam atstumo vienetui. Pateikiu skruzdžių kolonijos optimizavimo algoritmo pseudo-kodą [10]:

```
Inicializuoti Pėdsakus
Kartoti Kol (Sustojimo kriterijus netenkinamas) – Iteracijos Ciklas
{
      Kartoti Kol (Visos skruzdės baigia kelionę) – Kelionės Ciklas
      Lokalių Pėdsakų Atnaujinimas
      Baigti Kartoti
      Analizuoti Keliones
      Globalių Pėdsakų Atnaujinimas
}
```

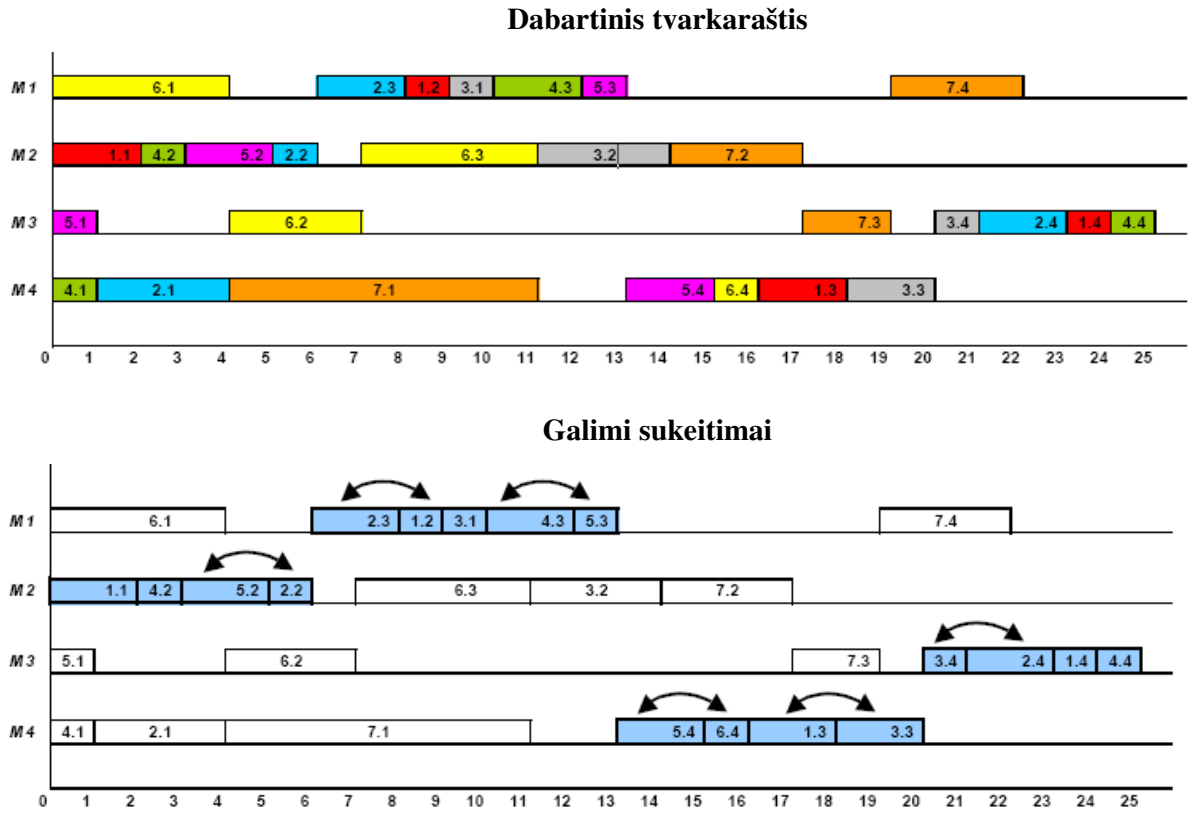
Feromonų išgaravimas – svarbi savybė norint išvengti perankstyto konvergavimo. Taip pat gali būti taikomi *demoniški veiksmai*, t.y. bet kokia lokali paieška rastiems sprendiniams.

Norint taikyti ACO algoritmą konkrečiam uždaviniui, sudaromas grafas atstovaujantis visus įmanomus skruzdžių kelius, t.y. jų eiliškumą. Šio grafo lankams ir suteikiami feromonų pėdsakai, naudojami ACO algoritme[10].

Sukurtoje programinėje įrangoje yra įgyvendintas ir modifikuotas skruzdžių algoritmas. Jei pasirenkama naudoti šį algoritmą, po visų veiksmų kurie standartiškai atliekami vykdant skruzdžių kolonijos optimizavimo algoritmą, atliekama ir vadinamoji lokalios paieškos procedūra. Tokiu būdu padedama algoritmui „ištrūkti“ iš lokalaus minimumo ir tokiu būdu nekonverguoti per anksti ir tokiu būdu leidžia pasiekti geresnę laiko kriterijaus reikšmę.

Kadangi nėra jokios garantijos, kad bet kuris tvarkaraštis, gautas konstravimo metu yra lokaliai optimalus, jam pritaikysime lokalią paiešką ir tokiu būdu pasistengsime sumažinti gamybos trukmę. Dėl šios priežasties mano naudojamas algoritmas vadinamas *genetiniu hibridiniu* (mišriuoju).

Naudosiu Novickio ir Smutnickio strategiją. Šios strategijos darbo objektas yra gretimų operacijų sekos, kitaip vaidinamos blokais (*kritinis kelias*)[2].



3.3 pav. Novickio ir Smutnickio strategija

Jei sukeitimas sutrumpina gamybos trukmę, jis yra išsaugojamas. Priešingu atveju sukeitimas yra anuliuojamas. Po sukeitimo gali atsirasti ir naujų blokų, juos taip pat reikia įvertinti. Jei sukeitimai nė karto nebepagerina gamybos laiko, lokalioji paieška nutraukiama.

Pateikiu lokalios paieškos aprašymą, pseudo kodo pavidale [2]:

```
Lokali_Paieška ( EsamasSprendinys )
  Vykdyti
  {
     EsamasSprendinysAtnaujintas = netiesa
     Rasti EsamasSprendinys visus kritinius blokus
     kol yra neapdirbtų blokų ir nėra EsamasSprendinysAtnaujintas vykdyti
     {
        jei nėra pimasis kritinis blokas tada
        {
           NaujasSprendinys:= EsamasSprendinys sukeisti bloko pirmas dvi operacijas
           {
           jei Trukmė (NaujasSprendinys ) < Trukmė (EsamasSprendinys ) tada
              EsamasSprendinys = NaujasSprendinys
              EsamasSprendinysAtnaujintas = tiesa
           }
        }
```

```
        jei nėra Paskutinis kritinis blokas ir nėra EsamasSprendinysAtnaujintas
        tada
           NaujasSprendinys = EsamasSprendinys sukeisti bloko paskutines operacijas
           jei GerumoFunkcija(NaujasSprendinys ) < GerumoFunkcija
           (EsamasSprendinys ) tada
           {
              EsamasSprendinys = NaujasSprendinys
              EsamasSprendinysAtnaujintas = tiesa
           }
        }
     }
  }
  kol EsamasSprendinysAtnaujintas = netiesa
Grąžinti EsamasSprendinys
```

PASIRINKTO UŽDAVINIO FORMULAVIMAS

Savo darbe pasirinkau nagrinėti JSSP tvarkaraščių sudarymo uždavinį. Kitaip tariant turi būti tenkinama sąlyga:

$$S \sim C \Leftrightarrow \forall i, x, y \ (\mathrm{x} < y \Rightarrow \mathrm{S}_{\mathrm{i,B_{i,x}}} \leq S_{i,B_{i,y}}) \qquad \textbf{(3.5)}$$

Čia matricos *S* ir *B* yra dydžio *J*M*. *B* apibrėžia seką, kuria turi būti vykdomas kiekvienas darbas, o sprendinys *S* apibrėžia kiekvienos operacijos vykdymo pradžios laiką. *C* tai uždavinio apribojimų rinkinys. Taip pat yra duota matrica *A*, kurioje apibrėžtos kiekvienos operacijos vykdymo trukmės. Šios matricos dydis taip pat *J*M*.

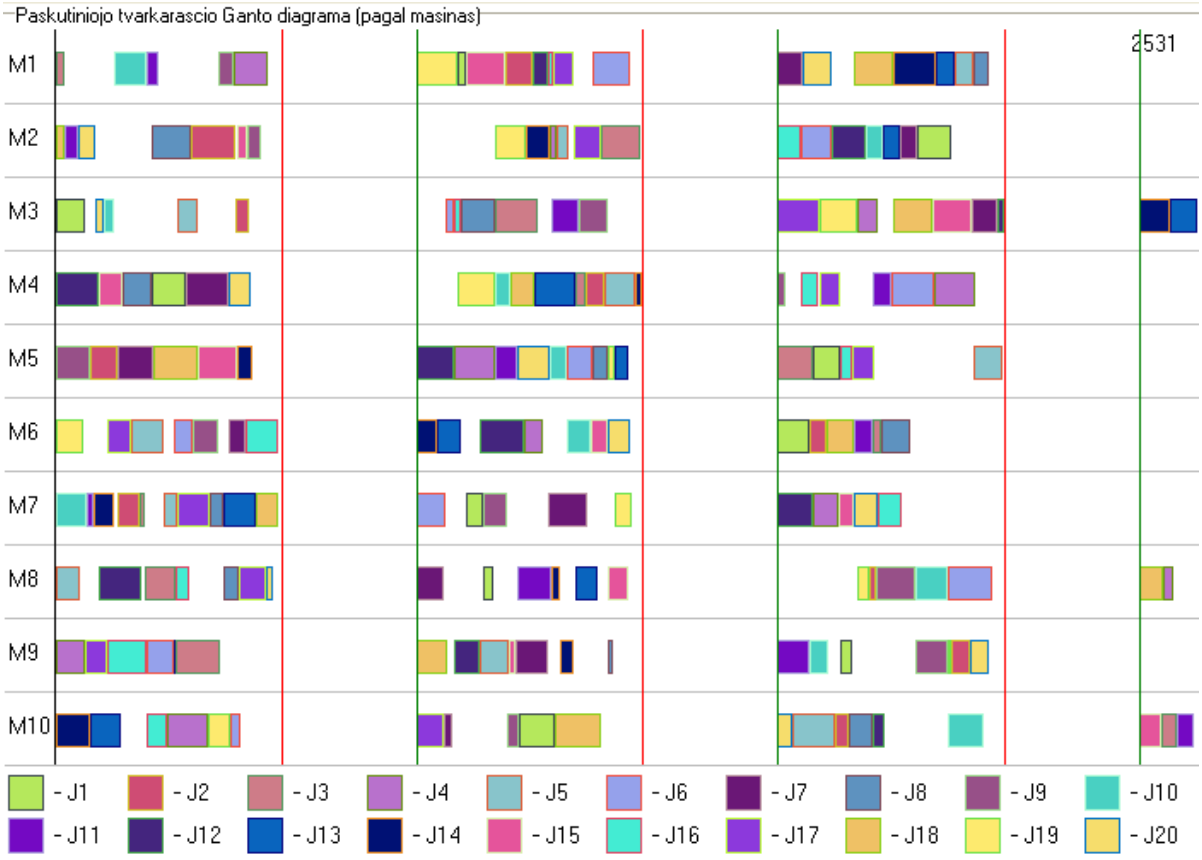
Matrica *A* įvedama iš failo. Eilutės reprezentuoja darbus, stulpeliai – mašinas. Gali būti įvedami tik sveikieji neneigiami skaičiai. Šiuo atveju 1 reprezentuoja mažiausią nedalomą gamybos trukmės vienetą.

Matrica *B* taip pat įvedama iš to paties failo po skirtuko „+“. Eilutės reprezentuoja darbus, stulpeliai operacijų eiliškumą (mažesnis stulpelio indeksas reiškia, kad operacija turi būti įvykdyta anksčiau). Gali būti įvedami tik sveikieji neneigiami skaičiai nuo 1 iki M ir eilutėse negali kartotis.

Šiuo atveju uždavinys šiek tiek panašus į FSSP uždavinį, nes visi darbai turi aplankyti mašinas tam tikru eiliškumu, tačiau tas eiliškumas skirtingiem darbam skiriasi. Vienos mašinos operacijas galima keisti vietomis. Kaip jau buvo aptarta skyrelyje *2.3*, šis uždavinys yra NP-sunkus, tai reiškia, kad jai spręsti yra reikalingas aproksimacinis algoritmas, pavyzdžiui toks kaip genetinis ar skruzdžių kolonijos optimizavimo.

Gamybos tvarkaraščių kriterijų *f*, kurį reikia minimizuoti , pasirinkau dažniausiai naudojamą tvarkaraščio darbo trukmę, t.y. laiką kada baigiama paskutinė operacija.

Be standartinių tvarkaraščių sudarymo uždavinio JSSP apribojimų, taip pat įgyvendinti ir papildomi darbo dienos apribojimai. Kadangi gamyklos ir gamybos linijos dažniausiai nedirba ištisą parą, todėl įtraukti ir nagrinėti tokius apribojimus tikslinga praktiniu požiūriu. Šių apribojimų analizė gali padėti išsiaiškinti gamybos proceso nutraukimo įtaką bendrai tvarkaraščio trukmei. Dėl dienos apribojimų optimalūs tvarkaraščiai gali gerokai skirtis nuo tų kuriems netaikomi šie apribojimai.



3.4 pav. Ganto diagramos pavyzdys su dienos apribojimais

Šiuo atveju gamybos laiko kriterijus toks (kur $D_{i,m}$ yra darbo dienų apribojimų matrica):

$$\mathrm{f(S)} = \max_{\mathrm{i,m}} (S_{i,m} + A_{i,m} + D_{i,m})$$

Darbo dienos apribojimai sudaryti iš darbo dienos pradžios ir darbo dienos pabaigos diskrečiųjų laikų. Šie laikai naudojami gerumo funkcijoje vertinant konkretaus tvarkaraščio trukmę. Nei viena operacija negali kirsti nei pradžios, nei pabaigos laiko. Jei operacijos neįmanoma įvykdyti nepažeidžiant šių reikalavimų, ji perkeliama į sekančią dieną.

TYRIMAI IR EKSPERIMENTAI

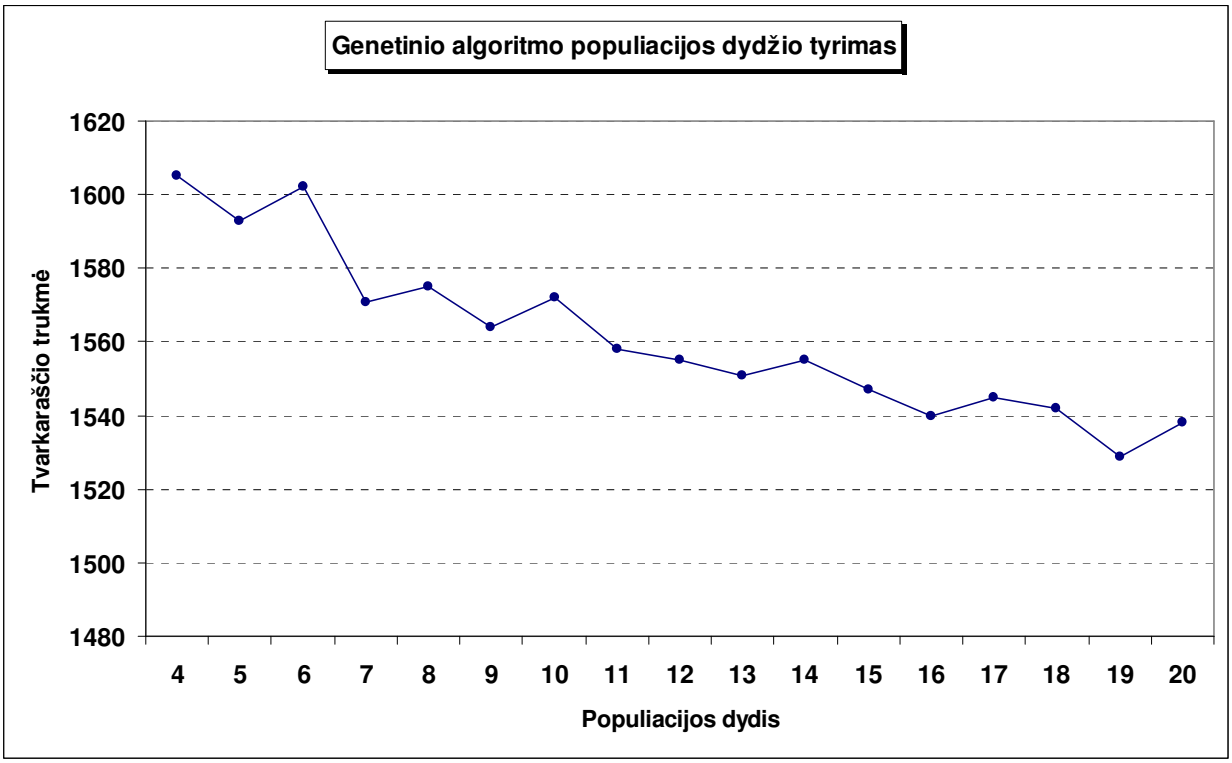
Šiame skyrelyje pateiksiu svarbiausių genetinio ir ACO algoritmo parametrų ir operatorių tyrimų prielaidas bei rezultatus.

GENETINIO ALGORITMO POPULIACIJOS DYDŽIO TYRIMAS

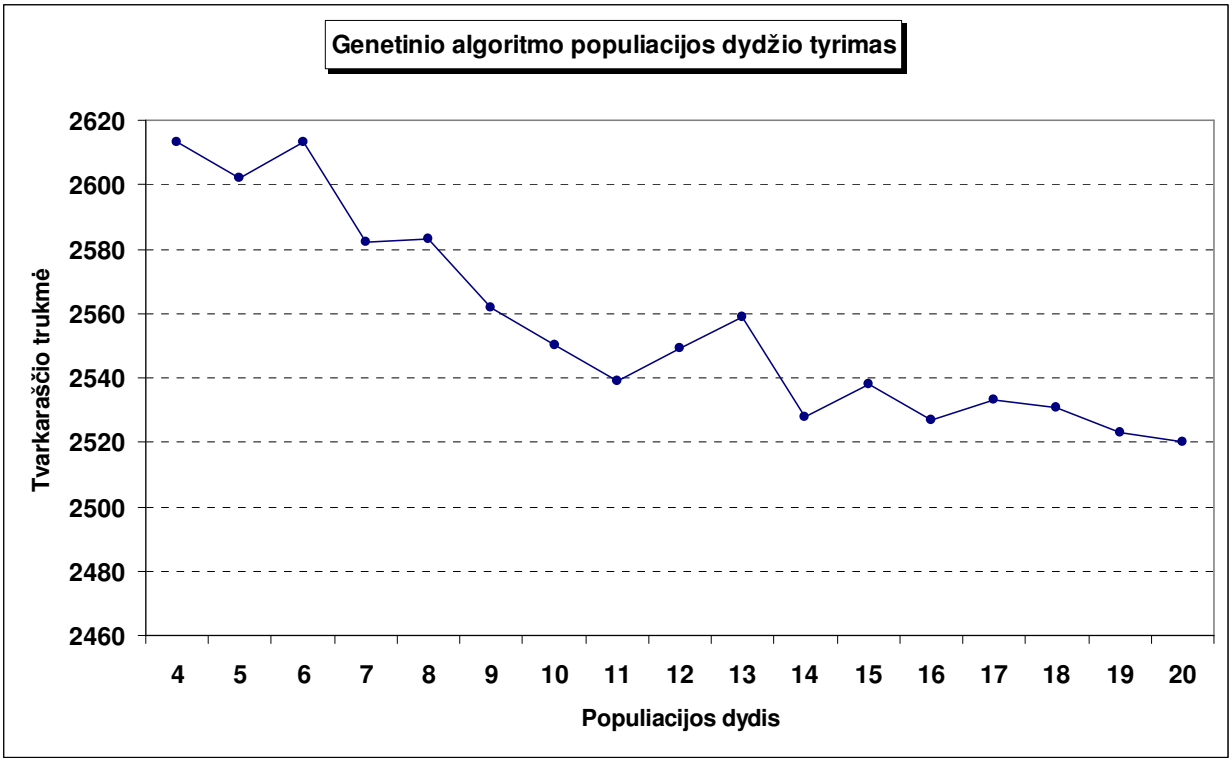
Populiacijos dydis yra labai svarbus parametras genetiniuose algoritmuose. Iš esmės populiacijos dydis apibrėžia, tai kiek genetinės informacijos apdorojama ir išsaugoma vienos iteracijos metu. Jei populiacija per maža, gali būti neišsaugoma pakankamai genetinės informacijos ir algoritmas gali funkcionuoti neefektyviai. Jei populiacija per didelė, algoritmas gali reikalauti daug kompiuterinių resursų, bet nepasiekti laukiamo rezultato pagerėjimo. Šiame skyrelyje tirsiu šio parametro svarbą.

Eksperimentai bus atliekami su duomenimis pateiktais *2 priede*, jų išmatavimai 20*10.

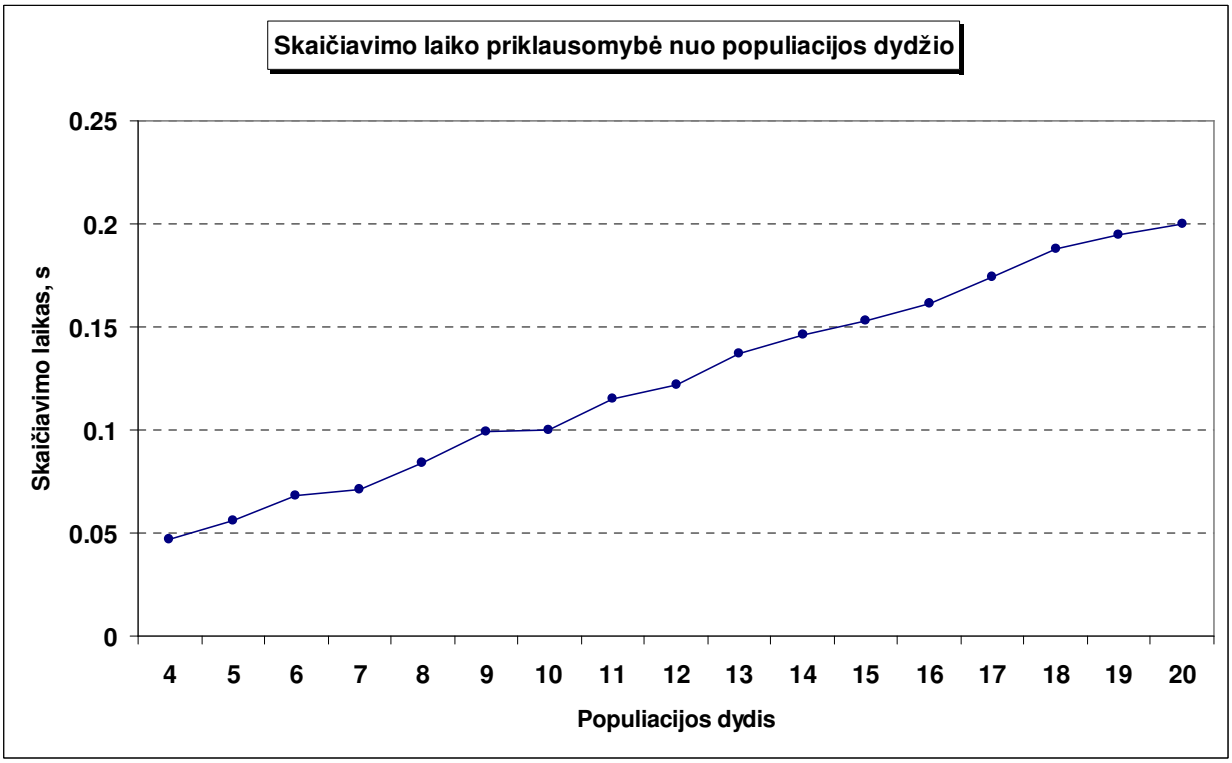
Genetinio algoritmo parametrai tokie: išliekančių individų 10%, mutuojančių individų 20%, vienataškis kryžminimas, 40 iteracijų.



3.5 pav. Gamybos trukmės priklausomybė nuo populiacijos dydžio



3.6 pav. Gamybos trukmės priklausomybė nuo populiacijos dydžio (su dienų apribojimais)



3.7 pav. Skaičiavimo laiko priklausomybė nuo populiacijos dydžio

Genetinio algoritmo be dienų apribojimų gauta gamybos trukmės kriterijaus reikšmė:

$$f(S) = \max_{i,m}(S_{GEN\,i,m} + A_{i,m}) = 1538 \quad \textbf{(3.6)}$$

Genetinio algoritmo su dienų apribojimais gauta gamybos trukmės kriterijaus reikšmė:

$$f_{dien}(S) = \max_{i,m}(S_{GEN\,i,m} + A_{i,m} + D_{i,m}) = 2520 \quad \textbf{(3.7)}$$

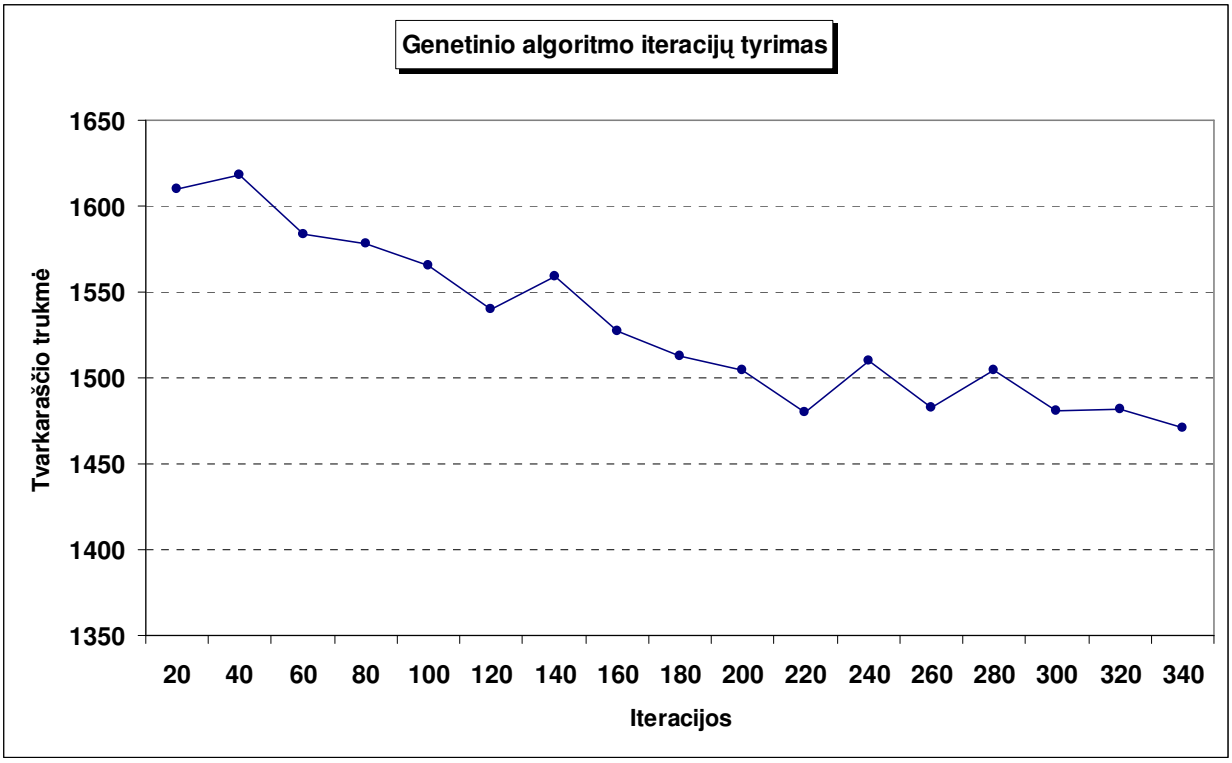
3.5 ir 3.6 paveiksluose kiekvienas taškas atstovauja penkių bandymų vidurkį. Iš gautų bandymų grafikų, (3.6) ir (3.7) galima teigti, kad optimalus populiacijos dydis uždaviniui be dienų apribojimų yra apie 15-17 individų ir 11-13 uždaviniui su dienų apribojimais. Įvertinus apytiksliai tiesinę skaičiavimo laiko priklausomybę nuo populiacijos dydžio, matome, kad naudoti didesnės populiacijos neverta, dėl didėjančio resursų sunaudojimo.

GENETINIO ALGORITMO ITERACIJŲ SKAIČIAUS TYRIMAS

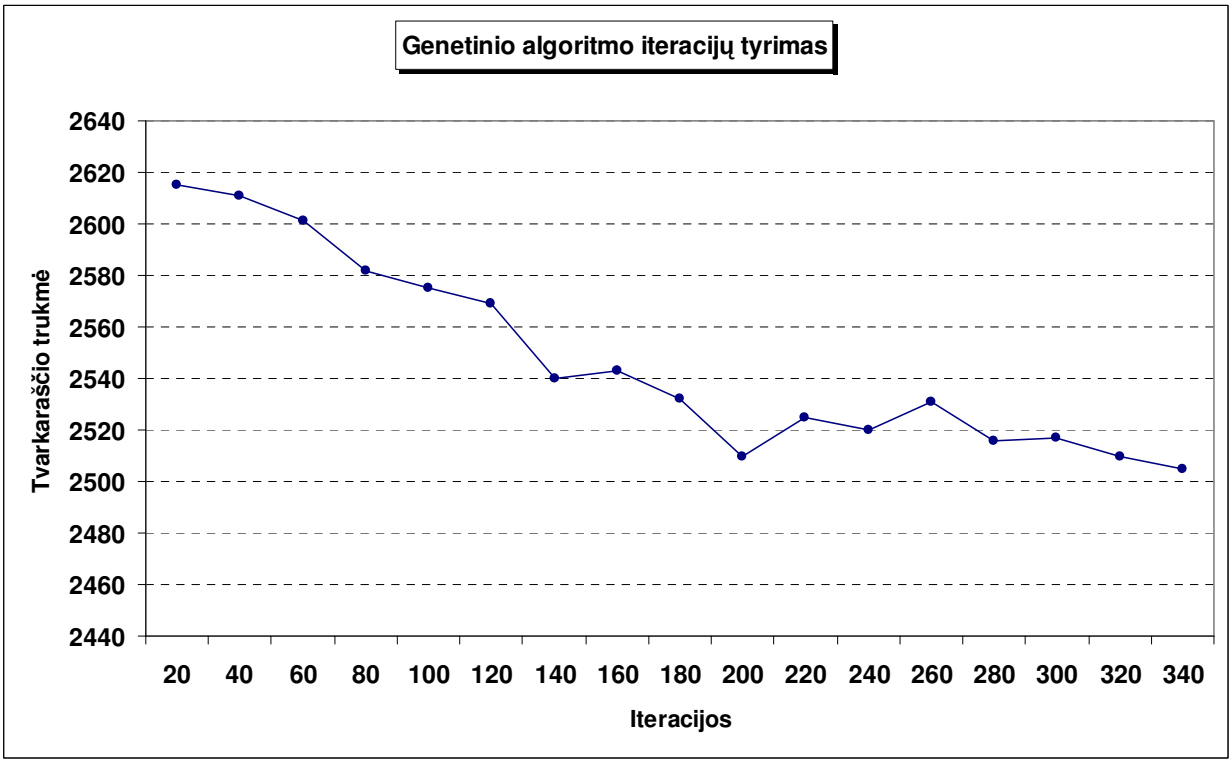
Kaip ir populiacijos dydis, iteracijų skaičius taipogi labai svarbus parametras bet kuriam optimizavimo algoritmui. Žinoma didesnis iteracijų skaičius padidina tikimybę rasti geresnį sprendinį, tačiau didesnis iteracijų skaičius taip pat pasireiškia ir didesniu resursų poreikiu. Taip pat jei algoritmas jau sukonvergavo, iteracijų skaičiaus didinimas yra tiesiog bevertis resursų švaistymas. Taigi šio skyrelio tyrimu panagrinėsiu iteracijų skaičiaus svarbą genetiniam algoritmui ir pasistengsiu rasti optimalią reikšmę.

Eksperimentai bus atliekami su duomenimis pateiktais *2 priede*, jų išmatavimai 20*10.

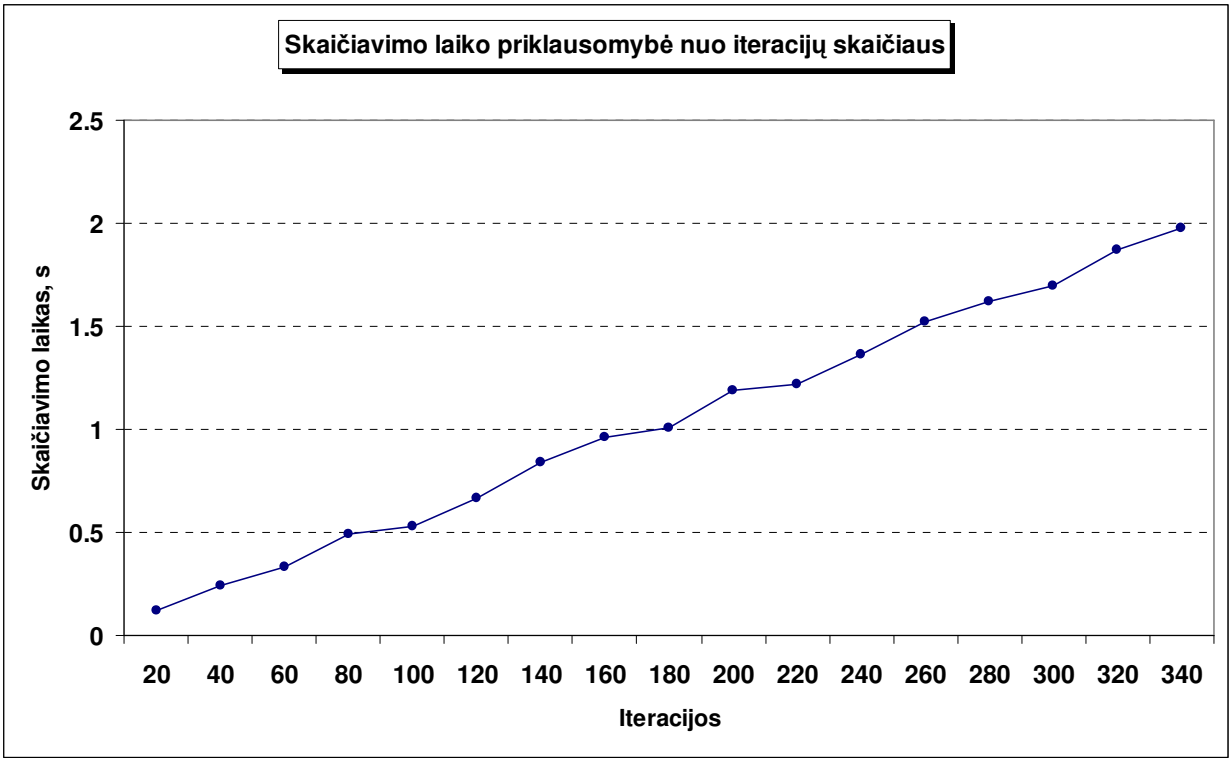
Genetinio algoritmo parametrai tokie: populiacijos dydis 20, išliekančių individų 4, mutuojančių individų 4.



3.8 pav. Gamybos trukmės priklausomybė nuo iteracijų skaičiaus



3.9 pav. Gamybos trukmės priklausomybė nuo iteracijų skaičiaus (su dienų apribojimais)



3.10 pav. Skaičiavimo laiko priklausomybė nuo iteracijų skaičiaus

Genetinio algoritmo be dienų apribojimų gauta gamybos trukmės kriterijaus reikšmė:

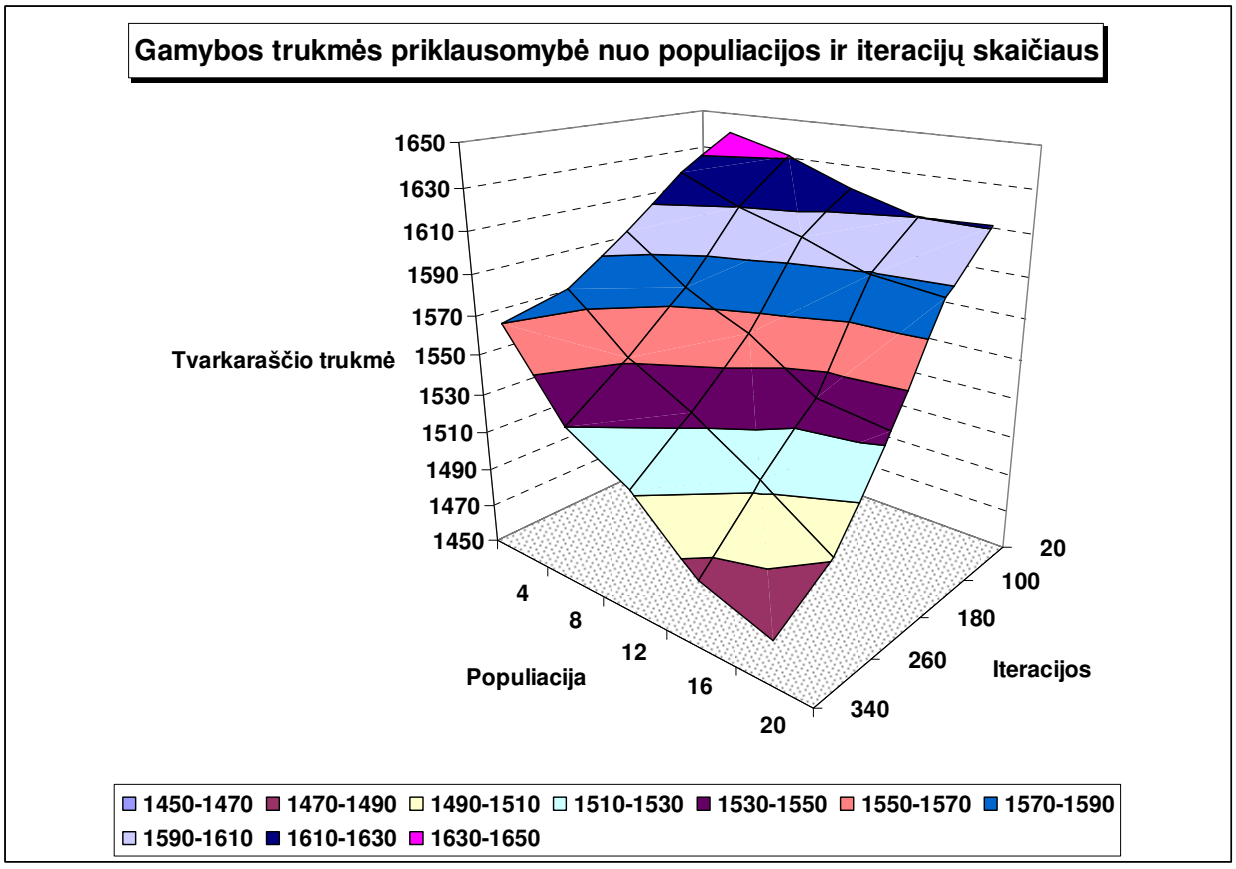
$$f(S) = \max_{i,m}(S_{GEN\,i,m} + A_{i,m}) = 1471 \qquad \textbf{(3.8)}$$

Genetinio algoritmo su dienų apribojimais gauta gamybos trukmės kriterijaus reikšmė:

$$f_{dien}(S) = \max_{i,m}(S_{GEN\,i,m} + A_{i,m} + D_{i,m}) = 2505 \qquad \textbf{(3.9)}$$

3.8 ir 3.9 paveiksluose kiekvienas taškas atstovauja penkių bandymų vidurkį. Iš gautų bandymų grafikų, (3.8) ir (3.9) galima teigti, kad optimalus iteracijų skaičius, spręsti mano pasirinktam konkrečiam uždaviniui, yra apie 160. Įvertinus apytiksliai tiesinę skaičiavimo laiko priklausomybę nuo iteracijų skaičiaus, matome, kad naudoti daugiau iteracijų neverta, dėl didėjančio resursų sunaudojimo.

Taigi, gavau panašias išvadas kaip ir skyrelyje 3.4.1 apie populiacijos dydį, t. y. abiejų šių parametrų kitimas minimizuojamąjį gamybos laiką veikia panašiai. Sekančiame paveiksle pavaizduoti rezultatai eksperimento, parodančio gamybos trukmės priklausomybę kartu nuo populiacijos dydžio ir iteracijų skaičiaus.

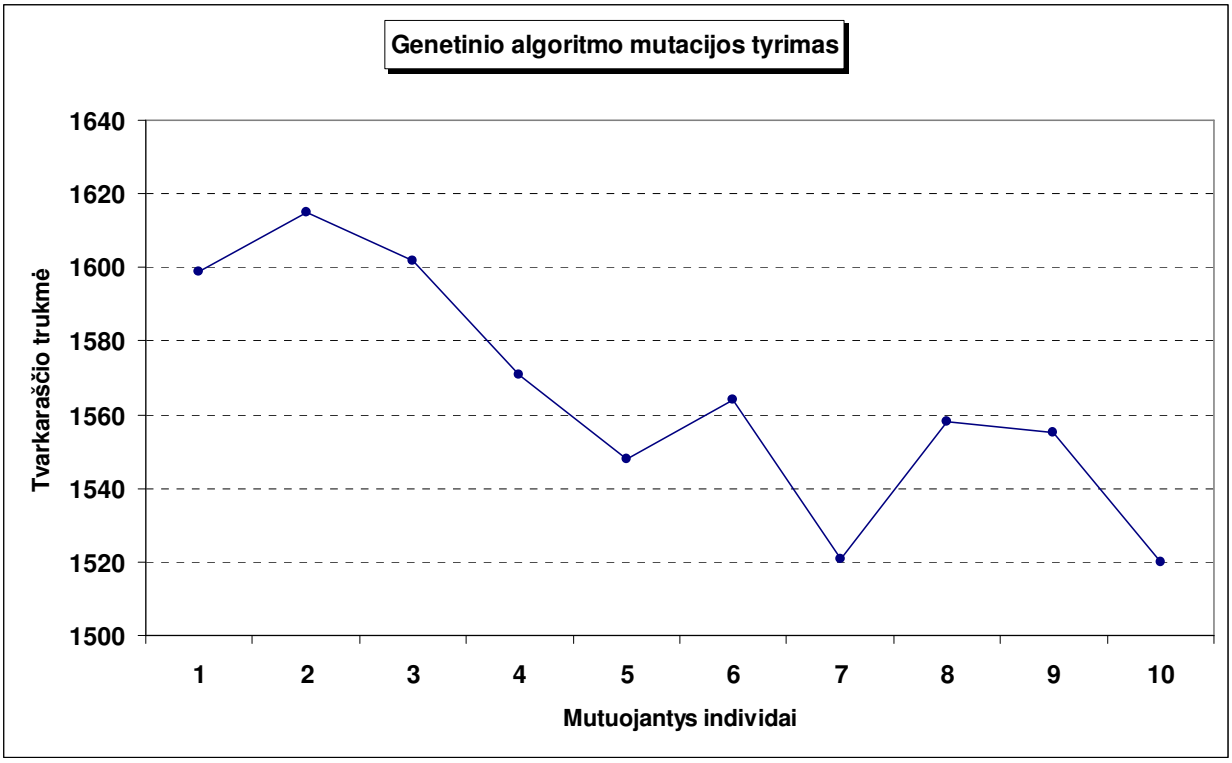


3.11 pav. Gamybos trukmės priklausomybė nuo populiacijos ir iteracijų skaičiaus

GENETINIO ALGORITMO MUTACIJOS TYRIMAS

Priminsiu, kad mano programoje naudojama, modifikuota mutacijos schema, kurioje išrenkami patys prasčiausi populiacijos individai, ir jie keičiami atsitiktinai sugeneruotais, o ne atsitiktinai pagal tam tikrą mažą tikimybę mutuojami populiacijos genai. Taip pat yra išsaugomas bent vienas populiacijos geriausias individas, kurio genai negali mutuoti (*elitinė strategija*). Tokiu būdu užtikrinama, kad algoritmas visada konverguos.

Ištirsiu kaip mutuojančių individų skaičiaus keitimas veikia gaunamą rezultatą, vykdant nedaug iteracijų. Genetinio algoritmo parametrai tokie: populiacijos dydis 13, išliekančių individų 1, vienataškis kryžminimas ir lokali paieška pradinei populiacijai.



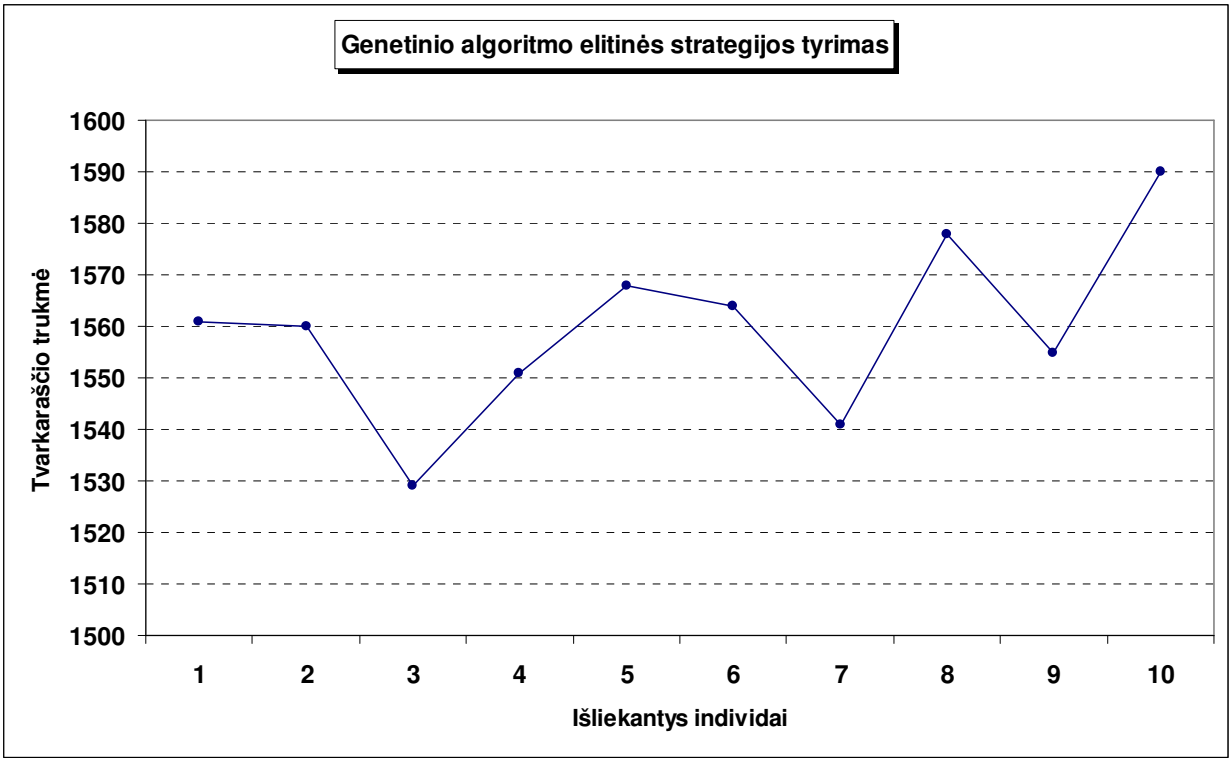
3.12 pav. Gamybos trukmės priklausomybė nuo mutuojančių individų skaičiaus

3.12 paveiksle kiekvienas taškas atstovauja penkių bandymų vidurkį. Iš gautų rezultatų matome, kad modifikuotoji mutacijos schema iš tiesų užtikrina metodo konvergavimą ir tuo pačiu, esant nedideliam iteracijų skaičiui, padidina geresnio tvarkaraščio radimo tikimybę. Matome, kad kai kuriais atvejais mutacijoje racionalu naudoti net iki pusės visos populiacijos. Tai aktualiau kai iteracijų skaičius nedidelis.

GENETINIO ALGORITMO ELITINĖS STRATEGIJOS TYRIMAS

Kaip buvo minėta anksčiau, mano naudojamas genetinis algoritmas naudoja elitinę strategiją. Šios strategijos esmė – geriausių populiacijos individų išsaugojimas ir perkėlimas į sekančią populiaciją. Šiuo tyrimu nagrinėsiu elitinės strategijos įtaką genetinio algoritmo optimizavimo rezultatams.

Genetinio algoritmo parametrai tokie: populiacijos dydis 15, mutojančių individų 3, vienataškis kryžminimas ir lokali paieška pradinei populiacijai.



3.13 pav. Elitinės strategijos tyrimas

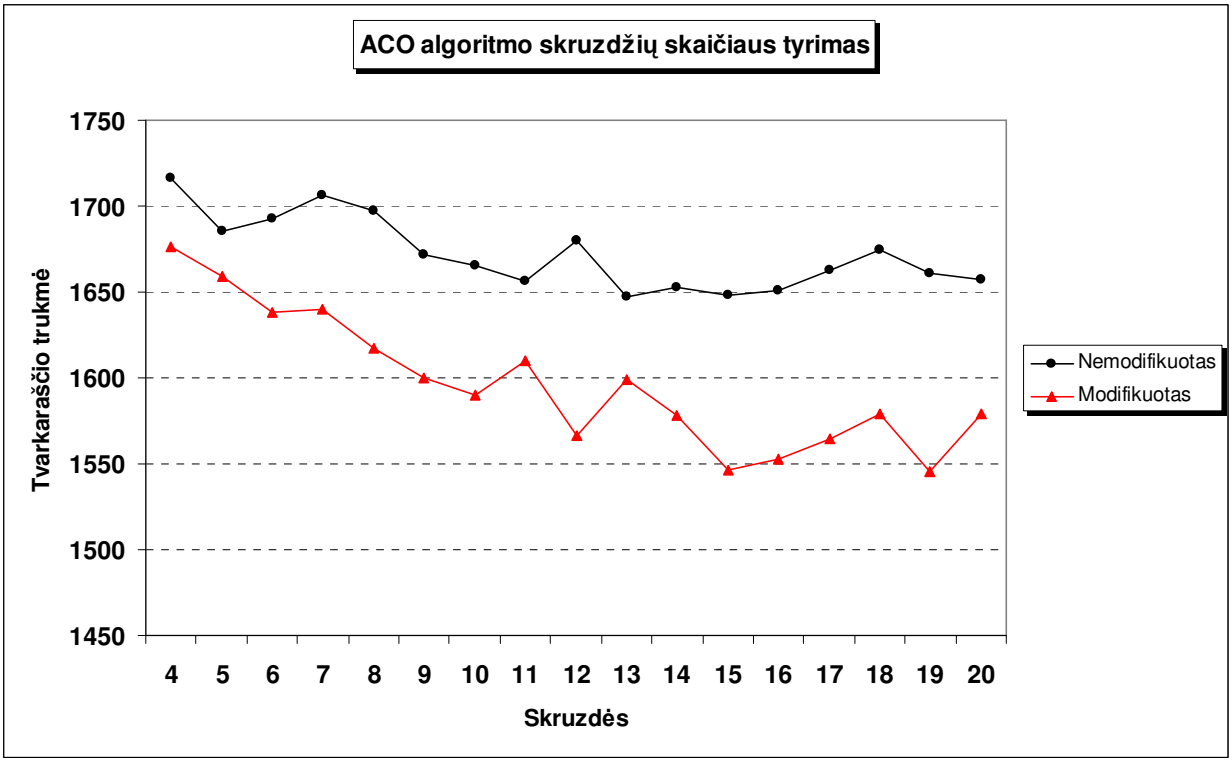
3.13 paveiksle kiekvienas taškas atstovauja penkių bandymų vidurkį. Iš gautų rezultatų matome, kad modifikuotoji mutacijos schema iš tiesų užtikrina metodo konvergavimą ir tuo pačiu, esant nedideliam iteracijų skaičiui, padidina geresnio tvarkaraščio radimo tikimybę. Matome, kad kai kuriais atvejais racionalu būtų naudoti net iki pusės visos populiacijos mutuojančius individus, t. y. tai aktualiau kai iteracijų skaičius nedidelis.

ACO ALGORITMO SKRUZDŽIŲ SKAIČIAUS TYRIMAS

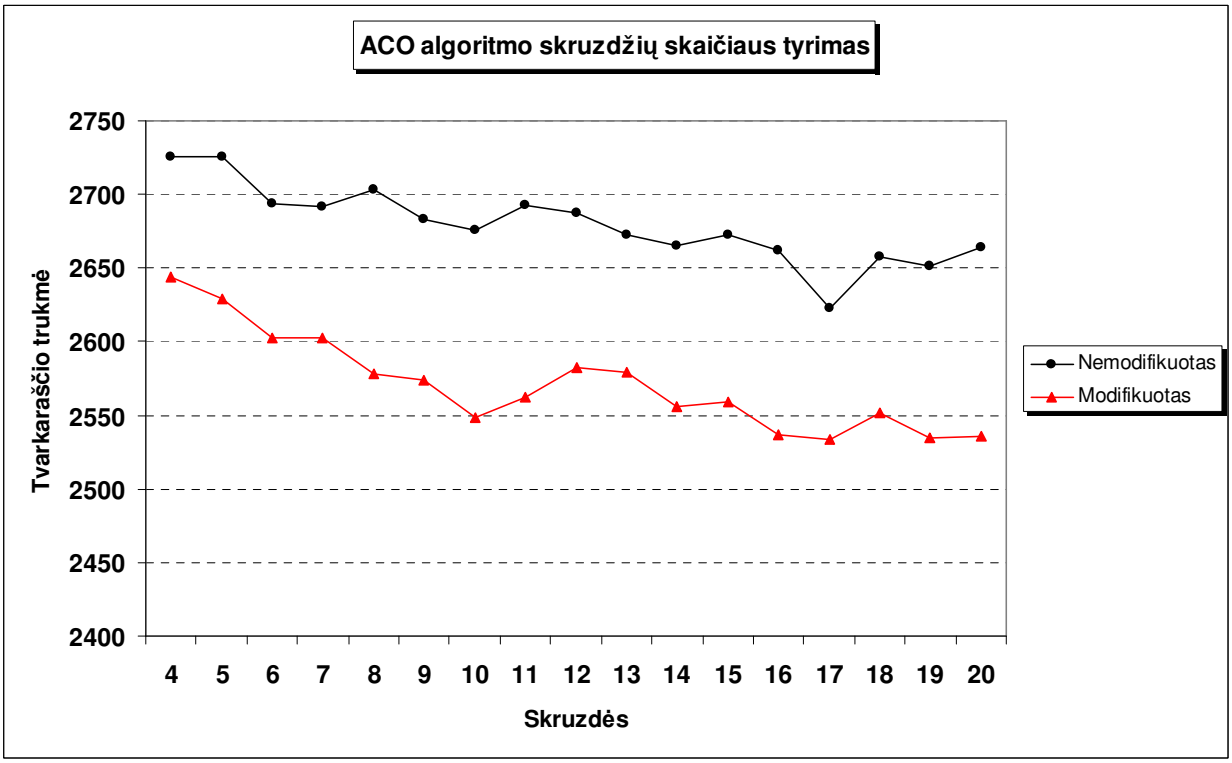
Skruzdžių skaičius yra labai svarbus parametras ACO algoritmuose. Iš esmės skruzdžių skaičius apibrėžia, tai kiek sruzdžių saugomos informacijos apdorojama ir išsaugoma vienos iteracijos metu. Jei skruzdžių per mažai, gali būti neišsaugoma pakankamai informacijos ir algoritmas gali funkcionuoti neefektyviai. Jei skruzdžių per daug, algoritmas gali reikalauti daug kompiuterinių resursų, bet nepasiekti laukiamo rezultato pagerėjimo. Šiame skyrelyje tirsiu šio parametro svarbą.

Eksperimentai bus atliekami su duomenimis pateiktais *2 priede*, jų išmatavimai 20*10.

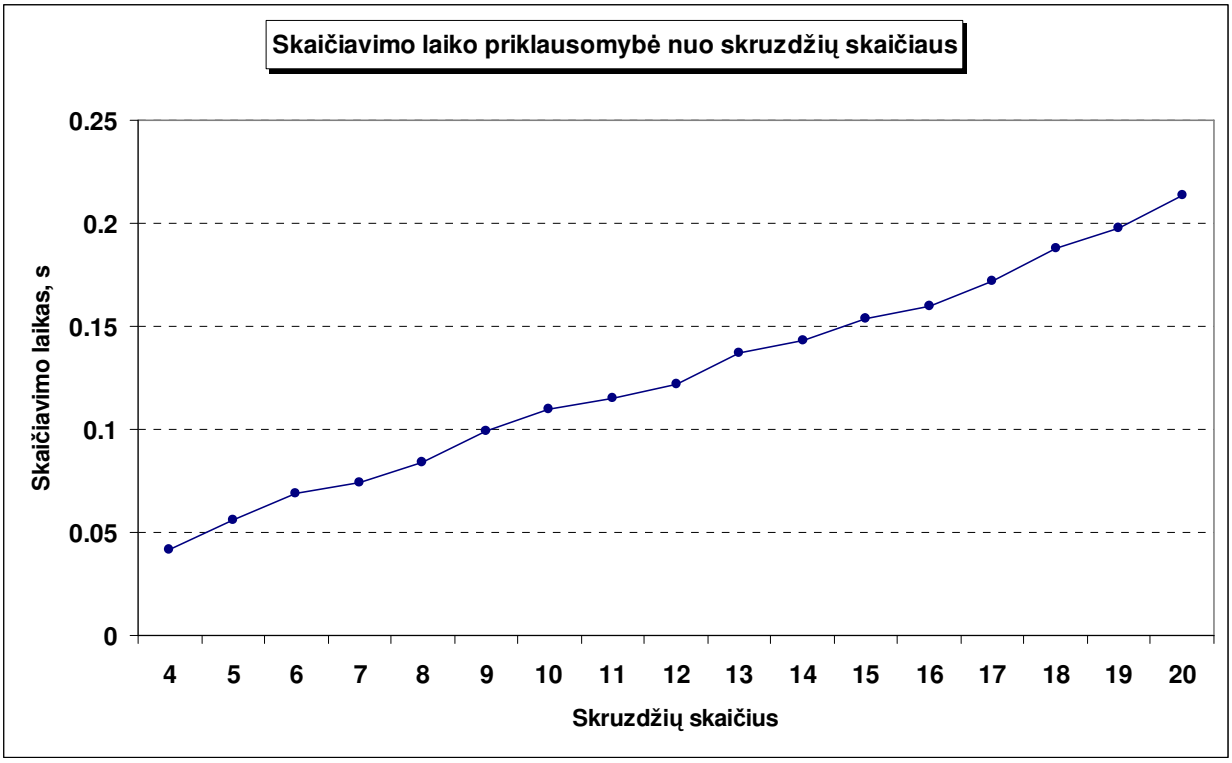
ACO algoritmo parametrai tokie: feromonų atnaujinimo koeficientas 0,02, papildomi pėdsakai, iteracijų 40.



3.14 pav. Gamybos trukmės priklausomybė nuo skruzdžių skaičiaus



3.15 pav. Gamybos trukmės priklausomybė nuo skruzdžių skaičiaus (su dienų apribojimais)



3.16 pav. Skaičiavimo laiko priklausomybė nuo populiacijos dydžio

Nemodifikuoto ACO be dienų apribojimų gauta gamybos trukmės kriterijaus reikšmė:

$$f(S) = \max_{i,m}(S_{ACO\,i,m} + A_{i,m}) = 1659 \qquad \textbf{(3.10)}$$

Modifikuoto ACO be dienų apribojimų gauta gamybos trukmės kriterijaus reikšmė:

$$f(S) = \max_{i,m}(S_{ACO\,\mathrm{mod}\,i,m} + A_{i,m} + D_{i,m}) = 1578 \qquad \textbf{(3.11)}$$

Nemodifikuoto ACO su dienų apribojimais gauta gamybos trukmės kriterijaus reikšmė:

$$f_{dien}(S) = \max_{i,m}(S_{ACO\,i,m} + A_{i,m}) = 2665 \qquad \textbf{(3.12)}$$

Modifikuoto ACO su dienų apribojimais gauta gamybos trukmės kriterijaus reikšmė:

$$f_{dien}(S) = \max_{i,m}(S_{ACO\,\mathrm{mod}\,i,m} + A_{i,m} + D_{i,m}) = 2536 \qquad \textbf{(3.13)}$$

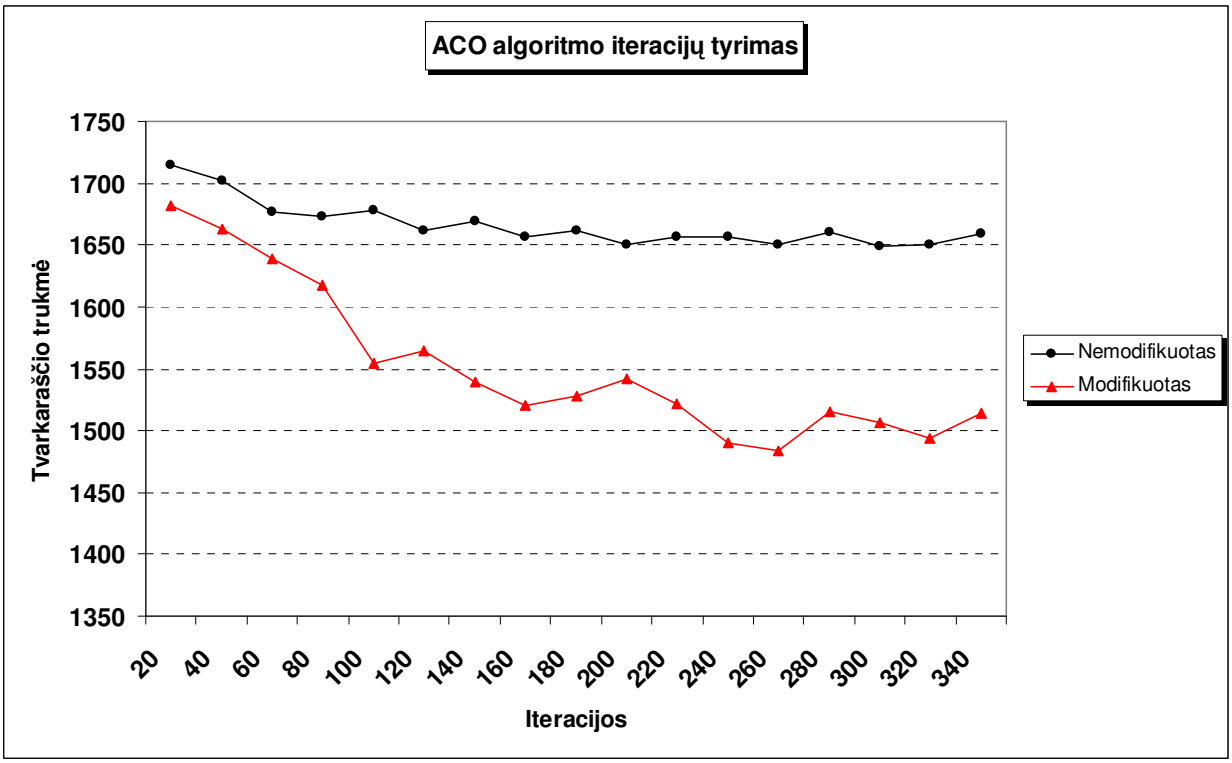
3.14 ir 3.15 paveiksluose kiekvienas taškas atstovauja penkių bandymų vidurkį. Iš gautų bandymų grafikų, (3.10) – (3.13) galima teigti, kad optimalus skruzdžių skaičius uždaviniui be dienų apribojimų yra apie 12-13 individų ir 15-16 uždaviniui su dienų apribojimais. Įvertinus apytiksliai tiesinę skaičiavimo laiko priklausomybę nuo skruzdžių skaičiaus, matome, kad naudoti daugiau skruzdžių neverta, dėl didėjančio resursų sunaudojimo. Akivaizdu, kad modifikuotasis ACO yra ženkliai pranašesnis esant bet kokioms skruzdžių skaičiaus reikšmėms.

ACO ALGORITMO ITERACIJŲ SKAIČIAUS TYRIMAS

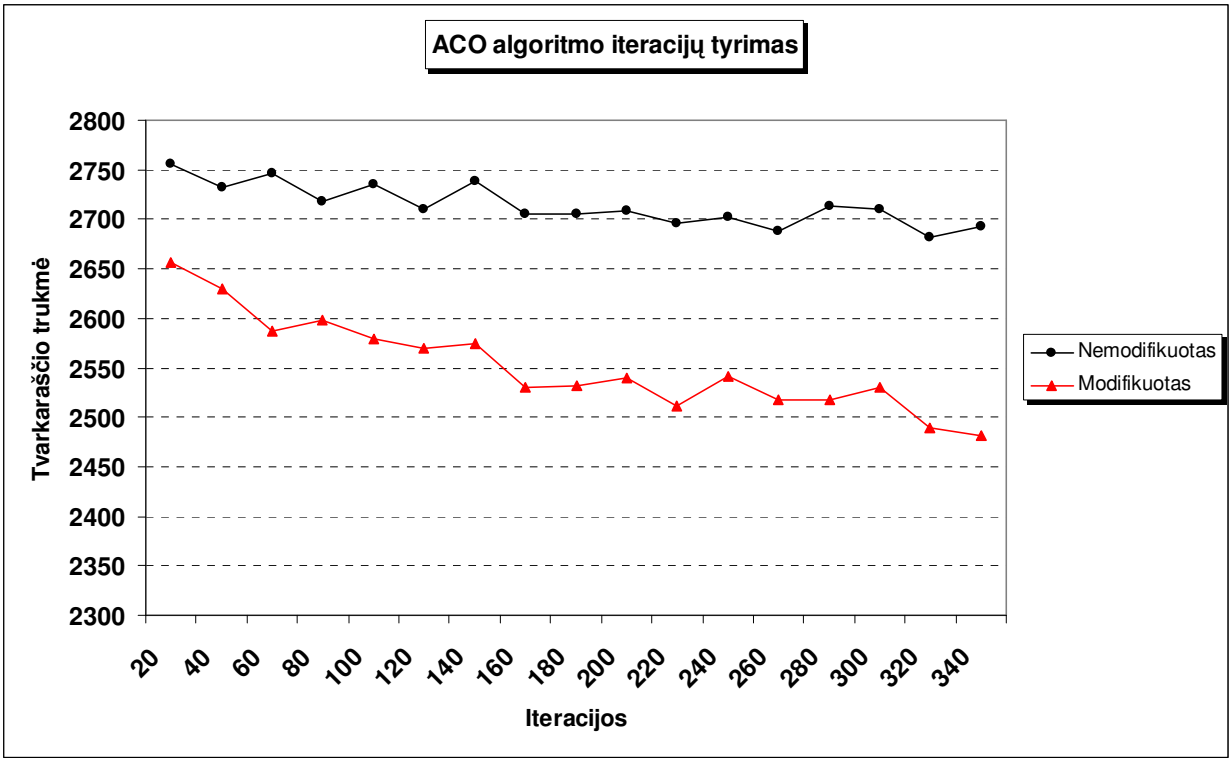
Kaip ir skruzdžių skaičius, iteracijų skaičius taipogi labai svarbus parametras bet kuriam optimizavimo algoritmui. Žinoma didesnis iteracijų skaičius padidina tikimybę rasti geresnį sprendinį, tačiau didesnis iteracijų skaičius taip pat pasireiškia ir didesniu resursų poreikiu. Taip pat jei algoritmas jau sukonvergavo, iteracijų skaičiaus didinimas yra tiesiog bevertis resursų švaistymas. Taigi šio skyrelio tyrimu panagrinėsiu iteracijų skaičiaus svarbą ACO algoritmui ir pasistengsiu rasti optimalią reikšmę.

Eksperimentai bus atliekami su duomenimis pateiktais *2 priede*, jų išmatavimai 20*10.

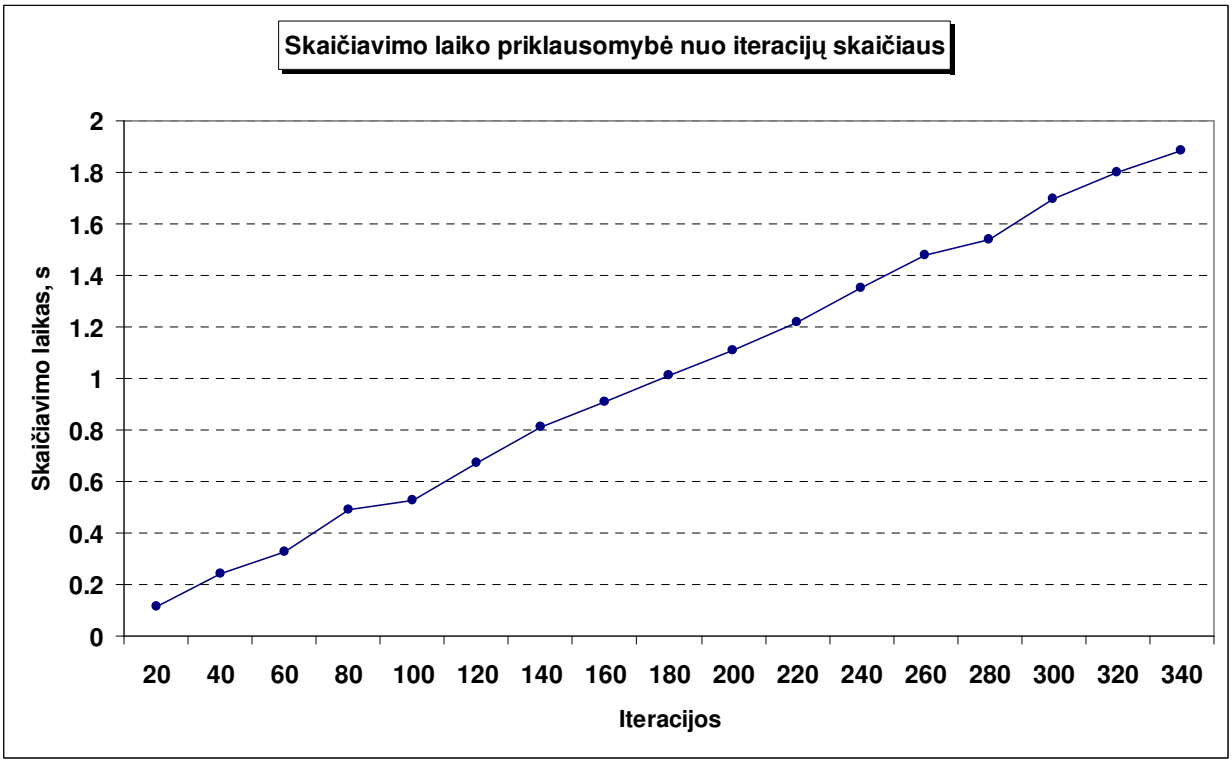
ACO algoritmo parametrai tokie: skruzdžių 10, feromonų atnaujinimo koeficientas 0,02, papildomi pėdsakai.



3.17 pav. Gamybos trukmės priklausomybė nuo iteracijų skaičiaus



3.18 pav. Gamybos trukmės priklausomybė nuo iteracijų skaičiaus (su dienų apribojimais)



3.19 pav. Skaičiavimo laiko priklausomybė nuo iteracijų skaičiaus

Nemodifikuoto ACO be dienų apribojimų gauta gamybos trukmės kriterijaus reikšmė:

$$\mathrm{f(S)} = \max_{\mathrm{i,m}} (S_{ACO\,i,m} + A_{i,m}) = 1661 \quad \textbf{(3.14)}$$

Modifikuoto ACO be dienų apribojimų gauta gamybos trukmės kriterijaus reikšmė:

$$\mathrm{f(S)} = \max_{\mathrm{i,m}} (S_{ACO\,\mathrm{mod}\,i,m} + A_{i,m} + D_{i,m}) = 1515 \quad \textbf{(3.15)}$$

Nemodifikuoto ACO su dienų apribojimais gauta gamybos trukmės kriterijaus reikšmė:

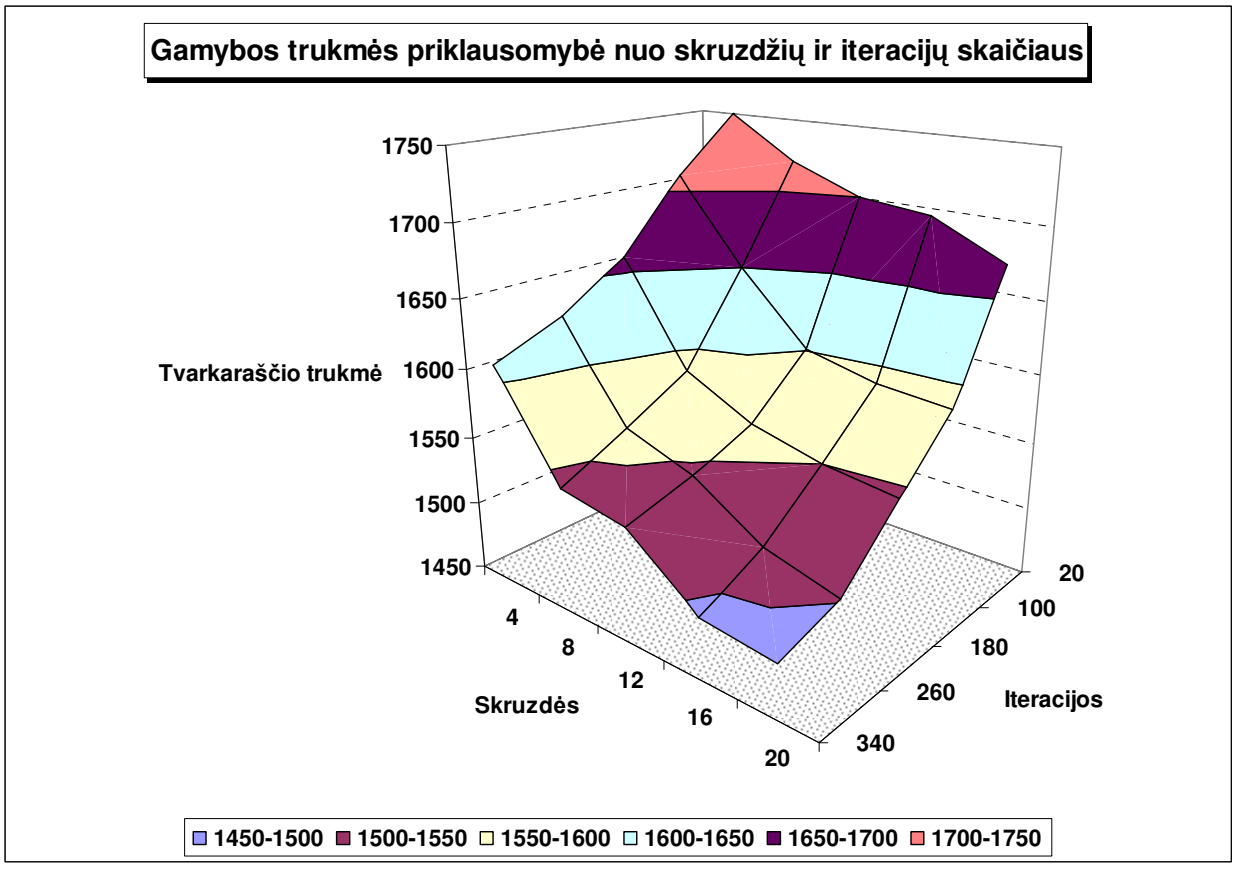
$$\mathrm{f_{dien}(S)} = \max_{\mathrm{i,m}} (S_{ACO\,i,m} + A_{i,m}) = 2696 \quad \textbf{(3.16)}$$

Modifikuoto ACO su dienų apribojimais gauta gamybos trukmės kriterijaus reikšmė:

$$\mathrm{f_{dien}(S)} = \max_{\mathrm{i,m}} (S_{ACO\,\mathrm{mod}\,i,m} + A_{i,m} + D_{i,m}) = 2471 \quad \textbf{(3.17)}$$

3.17 ir 3.18 paveiksluose kiekvienas taškas atstovauja penkių bandymų vidurkį. Iš gautų bandymų grafikų, (3.14) – (3.17) galima teigti, kad optimalus iteracijų skaičius, spręsti mano pasirinktam konkrečiam uždaviniui, yra apie 260. Įvertinus apytiksliai tiesinę skaičiavimo laiko priklausomybę nuo iteracijų skaičiaus, matome, kad naudoti daugiau iteracijų neverta, dėl didėjančio resursų sunaudojimo. Taip pat akivaizdu, kad modifikuotasis ACO yra ženkliai pranašesnis esant bet kokiam iteracijų skaičiui.

Taigi, gavau panašias išvadas kaip ir skyrelyje 3.4.5 apie populiacijos dydį, t. y. abiejų šių parametrų kitimas minimizuojamąjį gamybos laiką veikia panašiai. Sekančiame paveiksle pavaizduoti rezultatai eksperimento, parodančio gamybos trukmės priklausomybę kartu nuo populiacijos dydžio ir iteracijų skaičiaus.

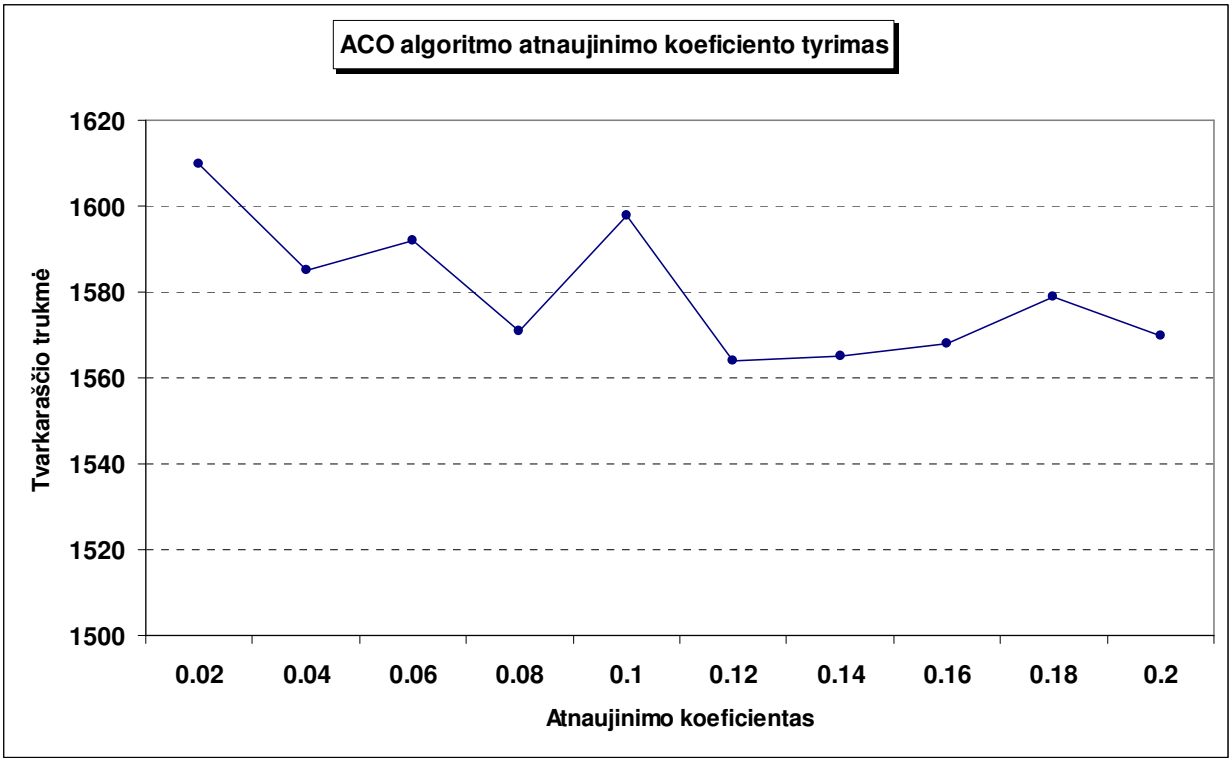


3.20 pav. Gamybos trukmės priklausomybė nuo populiacijos ir skruzdžių skaičiaus

ACO ALGORITMO ATNAUJINIMO KOEFICIENTO TYRIMAS

ACO algoritmo feromonų atnaujinimo koeficientas nustato kokiu greičiu atnaujinami skruzdžių feromonų pėdsakai, arba konkrečiu mano atveju feromonų matrica τ . Tai yra vienas iš svarbiausių ACO algoritmo parametrų. Jei parenkamos per mažos šio parametro reikšmės, algoritmas gali konverguoti lėčiau nei parinkus optimalią reikšmę. Tačiau pernelyg didelės koeficiento reikšmės parinkimas gali išprovokuoti ankstyvą algoritmo konvergavimą į lokalų minimumą.

Ištirsiu kaip atnaujinimo koeficiento keitimas veikia gaunamą rezultatą, vykdant nedaug iteracijų. ACO algoritmo parametrai tokie: skruzdžių 10, iteracijų 100, papildomi pėdsakai, 100 iteracijų.



3.21 pav. Gamybos trukmės priklausomybė nuo atnaujinimo koeficiento

3.21 paveiksle kiekvienas taškas atstovauja penkių bandymų vidurkį. Matome, kad šiuo konkrečiu atveju, optimali feromonų pėdsakų atnaujinimo koeficientų reikšmė yra maždaug 0,12, o didesnės reikšmės nebesuteikia rezultato pagerėjimo.

4. PROGRAMINĖ REALIZACIJA IR INSTRUKCIJA VARTOTOJUI

4.1 PROGRAMOS APRAŠYMAS

Programai rašyti pasirinkau C++ programavimo kalbą dėl to, kad tai kalba, su kuria teko plačiausiai susipažinti, taip pat ji turi pakankamai programinių galimybių pasirinkto uždavinio sprendimo įgyvendinimui. Programą rašiau atsižvelgdamas į tai, kad ji turi būti vartotojui patogi. Programai reikalinga operacinė sistema Windows 98/ME/2000/XP, taip pat pageidautinas pakankamai galingas kompiuteris, nes programos atliekamų skaičiavimų greitis tiesiogiai priklauso nuo jai prieinamų resursų. Sekančiuose skyreliuose trumpai aptarsiu svarbiausias mano programos savybes.

4.1.1 DUOMENŲ SKAITYMAS

Programa duomenis, t. y. matricas A ir B, nuskaito iš failo, kurio pavyzdžių galite rasti *1 ir 2 prieduose*. Kiekviena failo eilutė reprezentuoja matricų A ir B eilutę, t. y. vieną darbą su visų operacijų trukmėmis kiekvienai mašinai arba jų eiliškumą. Kiekvienas stulpelis reprezentuoja vieną mašiną. Faile sveikosios teigiamos reikšmės atskiriamos kableliais ir išsaugomos į failą su *.csv (angl. comma separated values)* plėtiniu. Tai yra daroma tam, kad failą būtų įmanoma atidaryti su Microsoft Office Excel programa. Tai yra patogu, norint susigeneruoti norimą reikšmių failą, patogu jį redaguoti. Norint atidaryti A ir B matricų duomenų failą spauskite mygtuką „Darbų duomenų failas“. Tuomet iš sąrašo išsirinkite duomenų failą *csv* formate.

Norint atidaryti dienų apribojimų duomenų failą spauskite mygtuką „Dienų duomenų failas“. Tuomet pasirinkite duomenų failą *csv* formate.

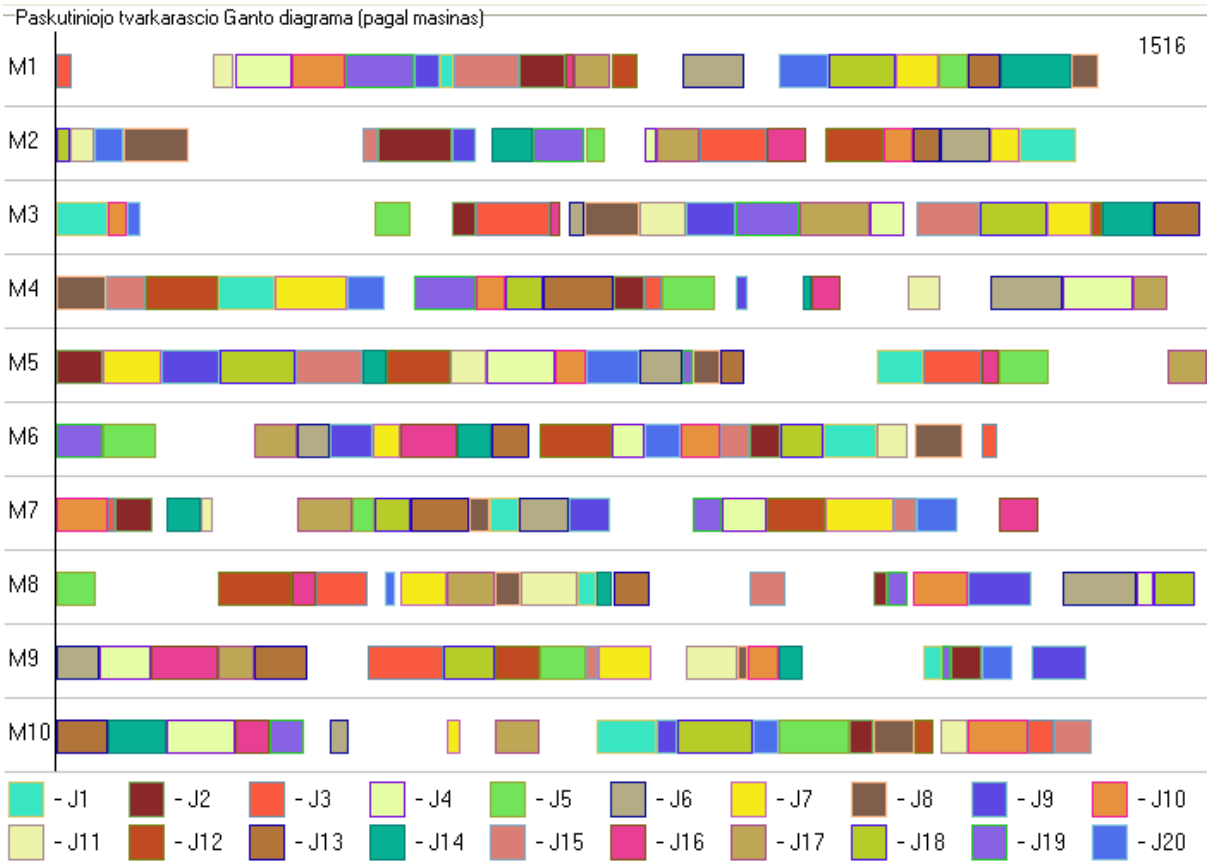
Atidarius norimą duomenų failą, programoje jį galima redaguoti, įterpti naują darbą arba ištrinti vieną iš esamų darbų, išsaugoti pakeitimus į kitą failą. Norėdami išvalyti pasirinktus duomenų failus, tam kad juos nuskaityti iš kito šaltinio, spauskite mygtuką „Išvalyti duomenis“. Kai pradiniai duomenys tenkina vartotoją, jis gali spausti mygtuką „Pasirinkti duomenis“ ir tokiu būdu duomenys galiausiai patvirtinami analizei ir sekančioje skiltyje jau įmanoma nustatyti įvairius parametrus ir pasirinkimus, kurie aptarti sekančiame skyrelyje.

4.1.2 PARAMETRAI IR KITI PASIRINKIMAI

Pasirinkus pradinius duomenis kuriuos norima analizuoti, sekančioje skiltyje galima rinktis optimizavimo algoritmus, t.y. genetinio, skruzdžių, skruzdžių modifikuoto. Tada galima nustatyti savo algoritmų parametrus arba palikti standartinius. Genetiniui algoritmui galima nustatyti populiacijos dydį, elitinių individų skaičių, generuojamų blogiausių individų skaičių ir mutacijos parametrus, t.y. individų skaičių ir genų skaičių. Skruzdžių algoritmui galima nustatyti skruzdžių skaičių, feromonų pėdsakų atnaujinimo koeficientą, skruzdžių skaičių jam, nustatyti ar naudoti papildomą pėdsakų išsaugojimo schemą ir nustatyti ar naudoti modifikuotą algoritmą, t.y. nustatyti paieškos gylio koeficientą didesnį nei nulis. Bent vieną kartą įvykužius optimizavimą, sekančioje skiltyje paskutinį tvarkaraštį galima atvaizduoti. Apie tai 4.1.3 skyrelyje.

4.1.3 VAIZDAVIMAS

Kaip ir daugeliu kitų atvejų, savo programoje tvarkaraščius pasirinkau vaizduoti Ganto diagramų pagalba. Sekančiame paveiksle pateikiu mano programos Ganto diagramos pagalba atvaizduotą tvarkaraštį (pagal duomenis iš *2 priedo*).



4.1 pav. Ganto diagramos pavyzdys

Kiekvienas darbas yra vaizduojamas skirtinga spalva, su skirtingu stačiakampio apvadu. Tai leidžia lengvai kiekvieną darbą atskirti vieną nuo kito. Mašinas reprezentuoja skirtingos eilutės. Kaip jau buvo minėta pirmajame skyriuje, grafinis tvarkaraščių naudojimas svarbus nustatant ar tvarkaraštis tenkina užsibrėžtus apribojimus ir tam, kad geriau suprasti, kaip realioje gamybos aplinkoje galėtų vykti gamyba.

Paveiksle pavaizduota Ganto diagrama yra pagal mašinas, t.y. viena eilutė rodo vienos mašinos gamybos procesas. Kad programoje matytumėte šį vaizdavimo variantą, spauskite trečiąją programos skiltį. Programa taip pat gali atlikti vaizdavimą ir pagal darbus, t.y. viena eilutė tada parodo vieno darbo gamybos procesą. Norint parodyti šį vaizdavimo variantą, spauskite ketvirtąją programos skiltį.

Abu aptarti vaizdavimai yra abipusiškai vienareikšmiai, nes iš esmės kitaip pateikia tą pačią informaciją.

5. DISKUSIJA

5.1 ALGORITMŲ PALYGINIMAS PAGAL ITERACIJAS

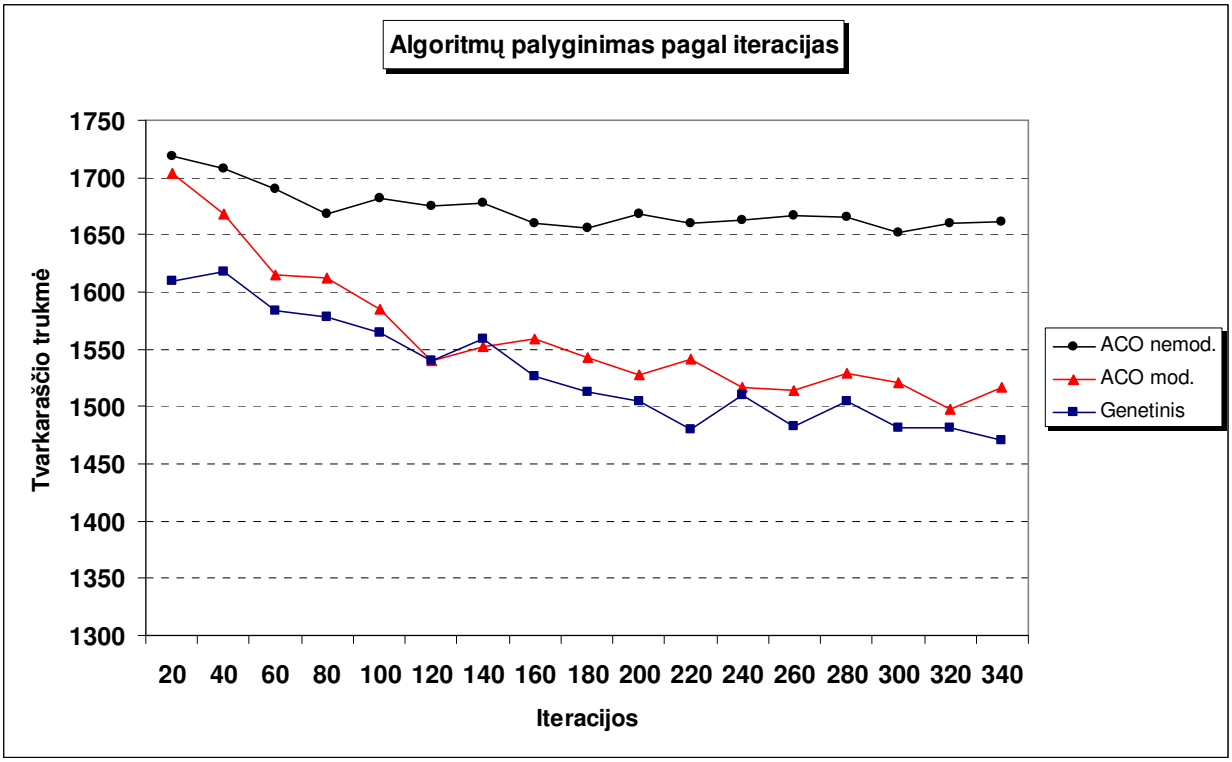
Iteracijų skaičius – svarbus parametras abiems nagrinėtiems algoritmams. Tiek ACO, tiek genetiniui algoritmui didesnis iteracijų skaičius leidžia tikėtis geresnio galutinio rezultato, arba konkrečiu mano atveju, mažesnės laiko kriterijaus reikšmės. Tačiau žinoma taip pat akivaizdu, kad padidėjęs iteracijų skaičius padidina ir kompiuterinės įrangos kuria vykdomi skaičiavimai resursų sunaudojimą. Kalbant dar tiksliau, didesnis iteracijų skaičius padidina procesoriaus laiką reikalingą atlikti algoritmų skaičiavimus.

Taigi, lyginant algoritmus, svarbu atsižvelgti į keletą jų veikimo aspektų. Vienas iš jų yra tai, kokią minimalią laiko kriterijaus reikšmę teoriškai gali pasiekti algoritmai. Tačiau svarbu ir tai kokiu greičių algoritmai konverguoja į jį. Algoritmas konverguojantis greičiau leidžia rasti geresnį konsensusą tarp iteracijų skaičiaus ir laiko kriterijaus reikšmės, nei algoritmas konverguojantis lėčiau. Tai leidžia naudoti mažiau iteracijų tam pačiam galutiniam rezultatui pasiekti, taigi tuo pačiu sutaupoma ir kompiuterinių resursų.

Eksperimentai bus atliekami su duomenimis pateiktais *2 priede*, jų išmatavimai 20*10.

Genetinio algoritmo parametrai tokie: populiacijos dydis 20, išliekančių individų 4, mutuojančių individų 4, daugiataškis kryžminimas.

ACO algoritmo parametrai tokie: skruzdžių 10, feromonų atnaujinimo koeficientas 0,02, papildomi pėdsakai.



5.1 pav. Algoritmų palyginimas pagal iteracijas

Iš grafiko matome, kad nemodifikuotas skruzdžių algoritmas konverguoja į lokalų minimumą ir gerokai nusileidžia efektyvumų pagal iteracijas kitiems algoritmams. Tačiau taip pat matome, kad algoritmo modifikavimas padeda ištrūkti iš šio „akligatvio“. Modifikuotas skruzdžių algoritmas jau stipriai nenusileidžią esant bet kokioms iteracijų reikšmėms. ACO tik kiek daugiau nusileidžia efektyvumu esant mažesnėms iteracijų reikšmėms.

Taigi lyginant algoritmus iteracijų prasme, genetinis algoritmas yra šiek tiek pranašesnis. Atsižvelgiant į tai, kad genetinio ir skruzdžių algoritmo resursų sunaudojimas yra panašus, šiuo atveju būtent genetinis algoritmas būtų optimaliausias pasirinkimas.

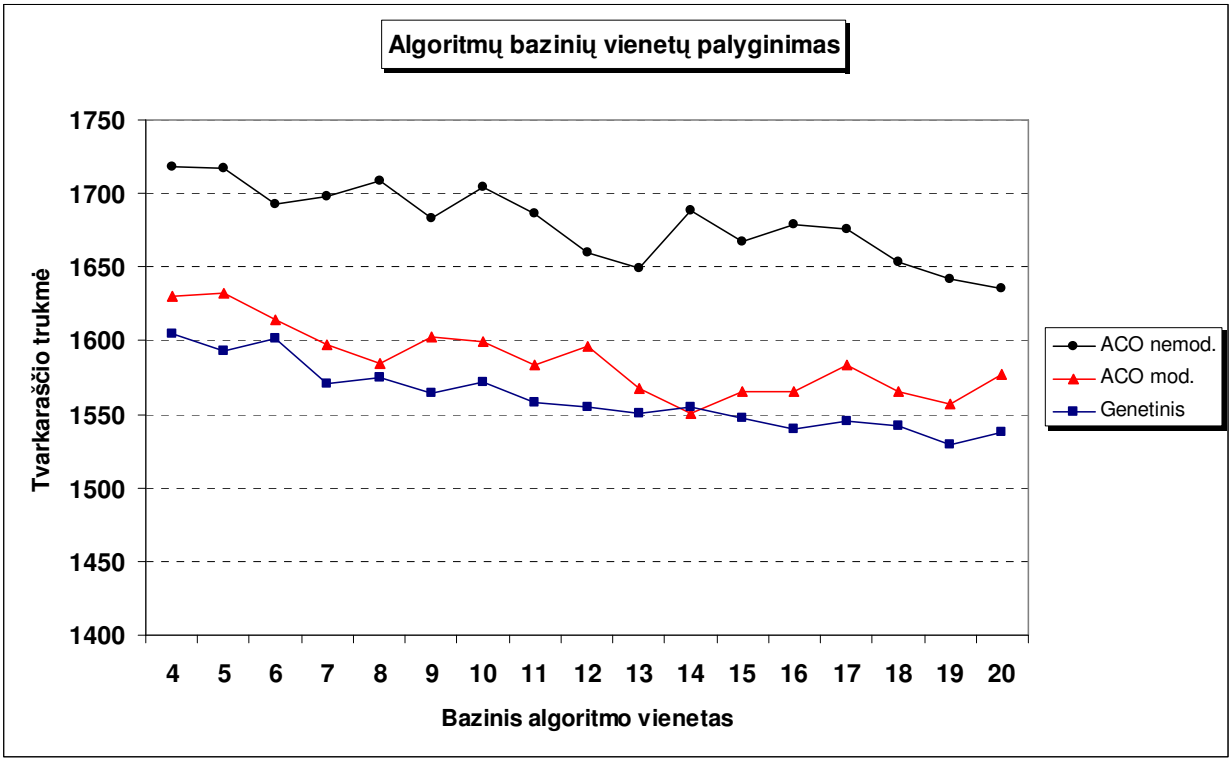
5.2 ALGORITMŲ BAZINIŲ VIENETŲ PALYGINIMAS

Kaip jau buvo minėta anksčiau, individų skaičius genetiniui algoritmui ir skruzdžių skaičius ACO algoritmui, kaip ir iteracijų skaičius, taip pat yra labai svarbus parametras. Šie dydžiai abiems algoritmams yra jų baziniai vienetai. Dėl šios priežasties skruzdžių skaičiaus ir populiacijos dydžio kitimas turi panašią įtaką gamybos laiko kriterijaus reikšmės kitimui – abiejų šių parametrų didinimas leidžia tikėtis mažesnės kriterijaus reikšmės, taigi tuo pačiu ir geresnio galinio rezultato. Be abejo šių

parametrų reikšmių didinimas tuo pačiu didina ir kompiuterinės įrangos resursų sunaudojimą, panašiai kaip ir iteracijų atveju.

Tačiau yra ir esminis skirtumas – iteracijų atveju beveik išimtinai didėja tik reikalingas skaičiavimam procesoriaus laikas, o skruzdžių skaičiaus ir populiacijos atveju didėja ir reikalingas procesoriaus laikas, ir programines įrangos naudojamas kompiuterio atminties kiekis. Taip yra todėl, kad ir skruzdės, ir individai nuo iteracijos prie iteracijos perneša savyje tam tikrą kiekį užkoduotos informacijos, o tam ir reikalingi atminties resursai. Dėl to bazinių vienetų pasirinkimas yra dar svarbesnis.

Taigi kaip ir iteracijų atveju, lyginant algoritmus, svarbu atsižvelgti į keletą jų veikimo aspektų: kokią minimalią laiko kriterijaus reikšmę teoriškai gali pasiekti algoritmai ir kokiu greičių algoritmai konverguoja į jį. Algoritmas konverguojantis greičiau leidžia rasti geresnį konsensusą tarp iteracijų skaičiaus ir laiko kriterijaus reikšmės, nei algoritmas konverguojantis lėčiau.



5.2 pav. Algoritmų bazinių vienetų palyginimas

Iš grafiko matome, kad kaip ir iteracijų atveju nemodifikuotas skruzdžių algoritmas konverguoja į lokalų minimumą ir gerokai nusileidžia efektyvumų pagal bazinius vienetus kitiems algoritmams. Taip pat ACO algoritmo modifikavimas padeda gerokai pagerinti jo veikimą. Modifikuotas skruzdžių algoritmas daugmaž tolygiai, nors ir nedaug, visame pasirinktame bazinių vienetų intervale atsilieka nuo genetinio algoritmo gamybos laiko kriterijaus prasme.

Taigi lyginant algoritmus iteracijų prasme, genetinis algoritmas yra šiek tiek pranašesnis. Atsižvelgiant į tai, kad genetinio ir skruzdžių algoritmo resursų sunaudojimas yra panašus, šiuo atveju būtent genetinis algoritmas būtų optimalesnis pasirinkimas.

5.3 DIENŲ APRIBOJIMŲ ĮTAKA GAMYBOS LAIKO KRITERIJUI

Panagrinėsiu kokią įtaką tvarkaraščiam turi dienų apribojimai. Tiriant buvo naudojamas genetinis algoritmas. Genetinio algoritmo parametrai tokie: populiacijos dydis 20, išliekančių individų 4, mutuojančių individų 4, iteracijų 300. Variantui kai naudojami dienų apribojimai, skaičiuojamas modifikuotas gamybos trukmės kriterijus, t.y. trukmė modifikuojama pagal tai, kiek buvo sugaišta laiko atliekant gamybos pauzes. Jei šiuo atveju gamybos trukmės kriterijus nebūtų modifikuojamas tokiu būdu, jo lyginimas su kriterijumi be dienų apribojimų netektų prasmes.

Nenaudojant dienų apribojimų gauta tokia gamybos kriterijaus reikšmė (laiko vienetais):

$$\mathrm{f(S)} = \max_{\mathrm{i,m}} (S_{GEN\,i,m} + A_{i,m}) = 1495 \qquad \textbf{(5.1)}$$

Naudojant dienų apribojimus gauta tokia modifikuoto gamybos kriterijaus reikšmė (laiko vienetais):

$$\mathrm{f_{dien}(S)} = \max_{\mathrm{i,m}} (S_{GEN\,i,m} + A_{i,m}) = 1614 \qquad \textbf{(5.2)}$$

Iš (5.1) ir (5.2) matome, kad dienų apribojimų naudojimas neigiamai įtakoja gamybos kriterijaus reikšmę, t.y. naudojant dienos apribojimus kriterijaus reikšmė didėja arba kitaip tariant, mažėja tvarkaraščio tankis. Tokį tankio mažėjimą galima paaiškinti tuo, kad naudojant dienos apribojimus mažėja galimų operacijų išrikiavimo variantų skaičius. Taip pat turi įtakos ir tai, kad atsiranda tokių situacijų kai tarp paskutinės operacijos ir darbo dienos pabaigos lieka per maži tarpai, kad tilptų bent viena iš galimų operacijų. Dienų apribojimų įtaka mažėja mažėjant vidutinės operacijos ilgiui arba didėjant darbo dienų ilgiui apribojimuose. Taip yra todėl, kad šiuo atveju santykinai sumažėja ir tvarkaraščio tarpai, atsiradę dėl dienų apribojimų.

IŠVADOS

1. Tiek genetinio, tiek skruzdžių kolonijos optimizavimo algoritmo gaunamiems rezultatams baziniai vienetai turi panašų poveikį kaip ir iteracijų skaičius. Abiejų šių parametrų didinimas leidžia tikėtis geresnio rezultato padidėjusio resursų sunaudojimo kaina.

2. Genetinis algoritmas mano pasirinktam tvarkaraščių sudarymo uždaviniui daugeliu atvejų yra efektyvesnis nei skruzdžių kolonijos optimizavimo algoritmas. Skruzdžių kolonijos optimizavimo algoritmo efektyvumui pagerinti taip pat reikalinga papildoma euristika. Jei pageidaujama sprendinį rasti greitai, tikslinga naudoti genetinį algoritmą.

3. Darbo dienų apribojimai didina tvarkaraščio trukmės kriterijaus reikšmę net ir tuo atveju, kai jis modifikuojamas taip, kad kompensuotų nedarbo laiką, o algoritmai taikant darbo dienų apribojimus konverguoja lėčiau.

4. Egzistuoja tiesinis ryšys tarp nagrinėtų algoritmų bazinių vienetų ir skaičiavimų sunaudojamų resursų. Bazinių vienetų skaičiaus didėjimas skatina skaičiavimo laiko ir atminties resursų augimą . Tai būdinga ir iteracijų skaičiui, tačiau šis skatina tik skaičiavimo laiko didėjimą.

REKOMENDACIJOS

Nors darbe buvo išnagrinėta nemažai tvarkaraščių sudarymo uždavinio ir jo algoritmų aspektų, tačiau dar liko daug erdvės ir kitiems tyrimams. Papildomai dar galima būtų ištirti dar daugiau kombinatorinio optimizavimo algoritmų, tinkančių spręsti gamybinių tvarkaraščių sudarymo uždavinį, pavyzdžiui tokių kaip modeliuojamas atkaitinimas, tabu paieška ir t.t.

Taip pat tolimesniais tyrimais galima nagrinėti įvairias kitas modifikacijas, pavyzdžiui lygiagretų algoritmų įgyvendinimą, įvairius hibridinius algoritmus su skritingais algoritmus, sukurti ir ištirti dar neaptartas jų modifikacijas. Taip pat tirti uždavinius su įvairiais kitais apribojimais, nagrinėti kitokio tipo tvarkaraščių sudarymo uždavinius, pavyzdžiui tokius kaip OSSP ir kiti.

PADĖKOS

Dėkoju magistro darbo vadovui doc. dr. Narimantui Listopadskiui už pagalbą ir patarimus. Taip pat dėkoju Sauliui Lazaravičiui ir Jonui Pokštui už naudingas diskusijas kombinatorinio optimizavimo temomis.

LITERATŪRA

1. Van Otterloo S. Evolutionary Algorithms and Scheduling Problems, 2002. p. 7-32.
2. Gonçalves J. F., de Magalhães Mendes J. J., Resende M. G. C. A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem, 2002. p. 2-11.
3. Lin S.-C., Goodman E. D., Punch III W. F. Investigating Parallel Genetic Algorithms on Job Shop Scheduling Problems, 1999. p. 1-10.
4. Carter B., Park K. How good are genetic algorithms at finding large cliques: an experimental study, 1993. p. 4-10.
5. Kämäräinen O., Ek V., Nieminen K., Ruuth S. Large scale generalized resource constrained scheduling problems: A genetic algorithm approach, 2000. p. 2-8.
6. Ritchie G., Levine J. A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments, 2004. p. 1-2.
7. Jensen M. T., Hansen T. K. Robust solutions to Job Shop problems, 1999. p. 7.
8. Delisle P., Krajecki M., Gravel M., Gagné C. Parallel Implementation of an Ant Colony Optimization Metaheuristic with OpenMp, 2001. p. 7.
9. Ventresca M., Ombuki B.M. Ant Colony Optimization for Job Shop Scheduling Problem, Technical Report # CS-04-04, Brock University, 2004. 11 p.
10. Delisle P., Krajecki M., Gravel M., Gagné C. Parallel implementation of an ant colony optimization metaheuristic with openmp, Université du Québec à Chicoutimi, 2001. 7 p.
11. Barker T., von Haartman M. Ant Colony Optimization, IE 516 Spring, 2005. 29 p.
12. Maniezzo V., Gambardella L.M., de Luigi F. Ant Colony Optimization, 2001. 21p.
13. Simonavičius J., Listopadskis N. Gamybinių tvatkaraščių genetinio algoritmo tyrimas, Matematika ir matematikos dėstymas, 2006, 6p.
14. Simonavičius J., Listopadskis N. Gamybinių tvatkaraščių sudarymo uždavinys, TMSM, 2006, 2p.
15. Simonavičius J., Listopadskis N. Tvarkaraščio sudarymo algoritmo analizė, Lietuvos matematikų draugijos XLVII konferencija, 2006, 5p.

SANTRUMPŲ IR TERMINŲ ŽODYNAS

Gamybos tvarkaraščių sudarymas – tai uždavinys, kurio tikslas rasti optimalius gamybos tvarkaraščius tam tikrai gamybinei situacijai.

Mašina – nedalomas gamybos proceso vienetas, atliekantis priskirtas operacijas.

Operacija – nedalomas gamybos proceso veiksmas, atliekamas vienos mašinos vienam darbui.

Darbas – gamybinių veiksmų seka, siekiant iš anksto numatyto tikslo.

FSSP – tvarkaraščiai tenkinantys $S \sim C \Leftrightarrow \forall i, j, m, n \ (\mathrm{S_{i,m}} \le S_{j,m} \Rightarrow \mathrm{S_{i,n}} \le S_{j,n})$.

JSSP – tvarkaraščiai tenkinantys $S \sim C \Leftrightarrow \forall i, x, y \ (\mathrm{x} < y \Rightarrow \mathrm{S_{i,B_{i,x}}} \le S_{i,B_{i,y}})$.

OSSP – tvarkaraščiai kuriems netaikomi eiliškumo apribojimai.

Genetinis algoritmas – tai algoritmas naudojantis kryžminimo ir mutacijos operatorius, kuris vykdomas ciklišk ai iki tam tikro sustojimo kriterijaus. Gali turėti ir daugiau papildomų operatorių.

ACO algoritmas (*angl. ant colony optimization*) – tai algoritmas naudojantis svorių sistemą (feromonų pėdsakus) priimant optimizavimo sprendimus, kuris vykdomas ciklišk ai iki tam tikro sustojimo kriterijaus. Gali turėti ir daugiau papildomų operatorių.

Individas – nedalomas bazinis genetinio algoritmo vienetas.

Skruzdė – nedalomas bazinis ACO algoritmo vienetas.

Genas – nedalomas genetinio algoritmo informacijos kiekis.

Chromosoma – genetinio algoritmo genų rinkinys.

Elitinė strategija – tai genetinio algoritmo rūšis, kai išsaugomi geriausi iteracijos individai ir perkeliami į sekančią iteraciją.

PRIEDAI

1 PRIEDAS. DARBO APRIBOJIMŲ DUOMENYS

Darbo dienos pradžia:

0, 500, 1300, 2100, 2900, 3700, 4500, 5300, 6100, 6900, 7700, 8500, 9300, 10100, 10900, 11700, 12500, 13300, 14100, 14900, 15700;

Darbo dienos pabaiga:

0, 800, 1600, 2400, 3200, 4000, 4800, 5600, 6400, 7200, 8000, 8800, 9600, 10400, 11200, 12000, 12800, 13600, 14400, 15200, 16000;

2 PRIEDAS. PAGRINDINIAI DUOMENYS

1.1 lentelė

A matricos duomenys

Mašina / Darbas	1	2	3	4	5	6	7	8	9	10
1	17	74	68	75	63	69	38	24	25	81
2	61	98	32	41	62	41	49	18	41	32
3	21	87	97	22	77	20	9	69	98	33
4	74	15	47	92	92	43	58	22	67	91
5	40	25	47	70	65	69	30	54	63	93
6	83	65	19	95	55	43	66	98	58	26
7	55	38	60	95	78	35	90	62	70	17
8	36	86	74	67	35	65	29	35	13	53
9	34	32	67	16	77	58	53	84	72	27
10	71	39	23	37	39	51	70	73	42	79
11	26	33	60	42	47	42	17	75	69	37
12	35	77	12	95	84	96	78	97	58	26
13	43	37	62	92	32	50	75	49	69	68
14	92	53	69	12	32	47	45	20	31	78
15	87	21	84	53	89	39	31	47	15	50
16	11	53	14	37	23	74	52	31	90	45
17	47	58	93	46	51	56	73	62	47	60
18	90	19	87	51	98	57	48	55	69	100
19	91	69	82	82	15	63	38	26	10	46
20	65	37	19	48	72	48	54	15	41	33

1.2 lentelė

B matricos duomenys

Eiliškumas / Darbas	1	2	3	4	5	6	7	8	9	10
1	3	10	1	2	8	7	4	5	9	6
2	5	3	4	6	1	7	2	9	10	8
3	1	7	5	6	8	9	2	3	4	10
4	3	6	8	9	4	5	7	10	1	2
5	9	6	4	7	10	2	3	1	5	8
6	7	8	5	9	6	2	4	10	1	3
7	8	9	10	2	1	3	7	4	6	5
8	10	2	5	1	6	9	3	4	7	8
9	3	4	7	8	1	2	5	9	10	6
10	3	8	2	4	5	6	1	9	7	10
11	3	1	6	9	4	8	2	5	7	10
12	6	8	10	1	3	5	7	2	4	9
13	9	8	10	5	7	4	3	6	2	1
14	9	5	10	8	3	4	2	6	7	1
15	4	3	9	1	2	6	8	7	5	10
16	6	7	5	8	9	4	10	3	1	2
17	47	58	93	46	51	56	73	62	47	60
18	90	19	87	51	98	57	48	55	69	100
19	91	69	82	82	15	63	38	26	10	46
20	65	37	19	48	72	48	54	15	41	33

3 PRIEDAS. PROGRAMOS TEKSTAI

Unit1.h

```
//---------------------------------------------------------------------------

#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Menus.hpp>
#include <ComCtrls.hpp>
#include <Dialogs.hpp>
#include <ToolWin.hpp>
#include <ExtCtrls.hpp>
#include <Grids.hpp>
#include <Buttons.hpp>
#include "matrix.h"
#include "Optimizavimas.h"
#include "Vaizdavimas.h"
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:  // IDE-managed Components
        TMainMenu *MainMenu1;
        TMenuItem *Failas1;
```

```
TMenuItem *Atidaryti1;
TMenuItem *Iseiti1;
TMenuItem *Pagalba1;
TMenuItem *Apieprograma1;
TMenuItem *Kiapnaudotis1;
TMenuItem *N1;
TOpenDialog *OpenDialog1;
TSaveDialog *SaveDialog1;
TStatusBar *StatusBar1;
TPageControl *PageControl1;
TTabSheet *TabSheet1;
TTabSheet *TabSheet2;
TTabSheet *TabSheet3;
TGroupBox *GroupBox1;
TButton *Button1;
TButton *Button2;
TGroupBox *GroupBox2;
TPageControl *PageControl2;
TTabSheet *TabSheet4;
TTabSheet *TabSheet5;
TTabSheet *TabSheet6;
TStringGrid *StringGrid1;
TStringGrid *StringGrid2;
TStringGrid *StringGrid3;
TGroupBox *GroupBox3;
TGroupBox *GroupBox4;
TGroupBox *GroupBox5;
TGroupBox *GroupBox6;
TLabel *Label1;
TLabel *Label2;
TButton *Button3;
TProgressBar *ProgressBar1;
TButton *Button4;
TLabel *Label3;
TButton *Button5;
TLabel *Label4;
TStringGrid *StringGrid4;
TPaintBox *PaintBox1;
TLabel *Label5;
TLabel *Label6;
TTabSheet *TabSheet7;
TGroupBox *GroupBox7;
TGroupBox *GroupBox8;
TPaintBox *PaintBox2;
TButton *Button6;
TButton *Button7;
TEdit *Edit1;
TLabel *Label7;
TEdit *Edit2;
TEdit *Edit3;
TLabel *Label9;
TEdit *Edit4;
TLabel *Label10;
TRadioButton *RadioButton1;
TRadioButton *RadioButton2;
TEdit *Edit5;
TCheckBox *CheckBox1;
TEdit *Edit6;
TLabel *Label8;
TLabel *Label11;
TLabel *Label12;
TEdit *Edit7;
TLabel *Label13;
TEdit *Edit8;
TLabel *Label14;
TEdit *Edit9;
TLabel *Label15;
TEdit *Edit10;
TEdit *Edit11;
TLabel *Label16;
TButton *Button8;
void __fastcall Button1Click(TObject *Sender);
void __fastcall Button2Click(TObject *Sender);
void __fastcall Iseiti1Click(TObject *Sender);
void __fastcall Button3Click(TObject *Sender);
void __fastcall Button4Click(TObject *Sender);
void __fastcall Button5Click(TObject *Sender);
void __fastcall PaintBox1Paint(TObject *Sender);
void __fastcall TabSheet3Show(TObject *Sender);
void __fastcall TabSheet7Show(TObject *Sender);
```

```
        void __fastcall PaintBox2Paint(TObject *Sender);
        void __fastcall Button7Click(TObject *Sender);
        void __fastcall Button6Click(TObject *Sender);
        void __fastcall Button8Click(TObject *Sender);

private:      // User declarations
        AnsiString AtidarytiDarbus();
        AnsiString AtidarytiDienas();
        void SunumeruotiEilIrStulpT(TStringGrid *);
        void SunumeruotiEilIrStulpM(TStringGrid *);
        void SunumeruotiEilIrStulpE(TStringGrid *);
        void SunumeruotiEilIrStulpD(TStringGrid *);


        int J, M;
        matrix DO; // darbas-operacijos laikas
        matrix DM; // darbas-masinos nr
        matrix DE; // darbas-eiliskumas (pagal masinos nr is eiles)
        matrix DApr; // dienu apribojimai
        int AprS; // dienu apribojimu skaicius
        matrix T; //sprendinys
        Optimizavimas * Opti;
        Vaizdavimas * Vaizd;
        //matrix
public:       // User declarations
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

Unit1.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
 DO = matrix(1,1,10);
 DM = matrix(1,1,1);
 DE = matrix(1,1,1);
 T = matrix(1,1,1);
 DApr = matrix(2,1,0);
 J = 1;
 M = 1;
 AprS=1;
 Vaizd = new Vaizdavimas(PaintBox1,PaintBox2);
 Vaizd->DuomenysVaizdavimui(J, M, AprS, DO, DM, DE, T, DApr);
}
//---------------------------------------------------------------------------



void __fastcall TForm1::Button1Click(TObject *Sender)
{
 StatusBar1->SimpleText = AtidarytiDarbus();
 PageControl2->ActivePageIndex = 0;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
 StatusBar1->SimpleText = AtidarytiDienas();
 PageControl2->ActivePageIndex = 2;
}
//---------------------------------------------------------------------------
AnsiString TForm1::AtidarytiDarbus()
{
```

```
StringGrid1->RowCount = 2;
StringGrid1->ColCount = 2;
StringGrid1->Cells[1][1] = "";
StringGrid2->RowCount = 2;
StringGrid2->ColCount = 2;
StringGrid2->Cells[1][1] = "";
FILE *iFileHandle;
int m = 0;
int mm = 0;
int j = 1;
char ch = ' ';
AnsiString Sk;
int Ski;
if (OpenDialog1->Execute())
{
 try
 {
  iFileHandle = fopen(OpenDialog1->FileName.c_str(), "r");
  while ( ch != '\n' && ch != '+')
  {
   ch = fgetc(iFileHandle);
   if (ch != ',' && ch != '\n')
   {
    Sk = Sk + ch;
   }
   else
   {
    m++;
    Ski = StrToInt(Sk);
    StringGrid1->ColCount += 1;
    StringGrid1->Cells[m][1] = IntToStr(Ski);
    Sk = "";
   }
  }
  StringGrid1->ColCount -= 1;
  ch = ' ';
  while ( !feof(iFileHandle) && ch != 'ÿ' && ch != '+' )
  {
   j++;
   ch = ' ';
   mm = 0;
   StringGrid1->RowCount += 1;
   while ( ch != '\n' && ch != 'ÿ' && ch != '+')
   {
    ch = fgetc(iFileHandle);
    if ( (ch == 'ÿ' || ch == '+') && mm != m-1 ) break;
    if (ch != ',' && ch != '\n' && ch != 'ÿ' && ch != '+')
    {
     Sk = Sk + ch;
    }
    else
    {
     mm++;
     Ski = StrToInt(Sk);
     StringGrid1->Cells[mm][j] = IntToStr(Ski);
     Sk = "";
    }
   }
   if (mm != m)
   {
    StringGrid1->RowCount -= 1;
    break;
   }
  }
  SunumeruotiEilIrStulpT(StringGrid1);
  while ( ch != '\n' && ch != 'ÿ') ch = fgetc(iFileHandle);
  ch = ' ';
  m = 0;
  mm = 0;
  j = 1;
  while ( ch != '\n' && ch != 'ÿ')
  {
   ch = fgetc(iFileHandle);
   if (ch != ',' && ch != '\n')
   {
    Sk = Sk + ch;
   }
   else
   {
    m++;
```

```
     Ski = StrToInt(Sk);
     StringGrid2->ColCount += 1;
     StringGrid2->Cells[m][1] = IntToStr(Ski);
     Sk = "";
    }
   }
   StringGrid2->ColCount -= 1;
   ch = ' ';
   while ( !feof(iFileHandle) && ch != 'ÿ'  )
   {
    j++;
    ch = ' ';
    mm = 0;
    StringGrid2->RowCount += 1;
    while ( ch != '\n' && ch != 'ÿ')
    {
     ch = fgetc(iFileHandle);
     if ( ch == 'ÿ' && mm != m-1 ) break;
     if (ch != ',' && ch != '\n' && ch != 'ÿ')
     {
      Sk = Sk + ch;
     }
     else
     {
      mm++;
      Ski = StrToInt(Sk);
      StringGrid2->Cells[mm][j] = IntToStr(Ski);
      Sk = "";
     }
    }
    if (mm != m)
    {
     StringGrid2->RowCount -= 1;
     break;
    }
   }
   SunumeruotiEilIrStulpM(StringGrid2);
   fclose(iFileHandle);
   Button3->Enabled = true;
   return "Failas nuskaitytas :)";
  }
  catch(...)
  {
   fclose(iFileHandle);
   StringGrid1->ColCount = 2;
   StringGrid1->RowCount = 2;
   StringGrid1->Cells[1][1] = "";
   return "Blogi duomenys!!!";
  }
 }
}
//---------------------------------------------------------------------------
AnsiString TForm1::AtidarytiDienas()
{
 StringGrid3->RowCount = 3;
 StringGrid3->ColCount = 2;
 StringGrid3->Cells[1][1] = "";
 FILE *iFileHandle;
 int m = 0;
 char ch = ' ';
 AnsiString Sk;
 int Ski;
 if (OpenDialog1->Execute())
 {
  try
  {
   iFileHandle = fopen(OpenDialog1->FileName.c_str(), "r");
   while ( ch != '\n' )
   {
    ch = fgetc(iFileHandle);
    if (ch != ',' && ch != '\n')
    {
     Sk = Sk + ch;
    }
    else
    {
     m++;
     Ski = StrToInt(Sk);
     StringGrid3->ColCount += 1;
     StringGrid3->Cells[m][1] = IntToStr(Ski);
```

```
     Sk = "";
    }
   }
   StringGrid3->ColCount -= 1;
   ch = ' ';
   m = 0;
   while ( ch != '\n' )
   {
    ch = fgetc(iFileHandle);
    if (ch != ',' && ch != '\n')
    {
     Sk = Sk + ch;
    }
    else
    {
     m++;
     Ski = StrToInt(Sk);
     StringGrid3->Cells[m][2] = IntToStr(Ski);
     Sk = "";
    }
   }
   SunumeruotiEilIrStulpD(StringGrid3);
   fclose(iFileHandle);
   return "Failas nuskaitytas :)";
  }
  catch(...)
  {
   fclose(iFileHandle);
   StringGrid3->ColCount = 2;
   StringGrid3->RowCount = 3;
   StringGrid3->Cells[1][1] = "";
   return "Blogi duomenys!!!";
  }
 }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Iseiti1Click(TObject *Sender)
{
 Close();
}
//---------------------------------------------------------------------------
void TForm1::SunumeruotiEilIrStulpM(TStringGrid * Gridas)
{
 int m = Gridas->ColCount - 1;
 int j = Gridas->RowCount - 1;
 J = j;
 M = m;
 for (int i = 1; i <= j; i++)
  Gridas->Cells[0][i] = "J" + IntToStr(i);
 for (int k = 1; k <= m; k++)
  Gridas->Cells[k][0] = "M" + IntToStr(k);
 Gridas->FixedCols = 1;
 Gridas->FixedRows = 1;
}
//---------------------------------------------------------------------------
void TForm1::SunumeruotiEilIrStulpT(TStringGrid * Gridas)
{
 int m = Gridas->ColCount - 1;
 int j = Gridas->RowCount - 1;
 for (int i = 1; i <= j; i++)
  Gridas->Cells[0][i] = "J" + IntToStr(i);
 for (int k = 1; k <= m; k++)
  Gridas->Cells[k][0] = "T" + IntToStr(k);
 Gridas->FixedCols = 1;
 Gridas->FixedRows = 1;
}
//---------------------------------------------------------------------------
void TForm1::SunumeruotiEilIrStulpE(TStringGrid * Gridas)
{
 int m = Gridas->ColCount - 1;
 int j = Gridas->RowCount - 1;
 for (int i = 1; i <= j; i++)
  Gridas->Cells[0][i] = "J" + IntToStr(i);
 for (int k = 1; k <= m; k++)
  Gridas->Cells[k][0] = "E" + IntToStr(k);
 Gridas->FixedCols = 1;
 Gridas->FixedRows = 1;
}
//---------------------------------------------------------------------------
void TForm1::SunumeruotiEilIrStulpD(TStringGrid * Gridas)
```

```
{
 int m = Gridas->ColCount - 1;
 AprS = m;
 Gridas->Cells[0][1] = "Pabaiga";
 Gridas->Cells[0][2] = "Pradzia";
 for (int k = 1; k <= m; k++)
  Gridas->Cells[k][0] = "D" + IntToStr(k);
 Gridas->FixedCols = 1;
 Gridas->FixedRows = 1;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
 DO = matrix(J,M,0);
 int m = M;
 int j = J;
 int da = AprS;
 int i = 0;
 int k = 0;
 for (i = 1; i <= j; i++)
 {
  for (k = 1; k <= m; k++)
  {
   DO(i,k) = StrToInt(StringGrid1->Cells[k][i]);
  }
 }
 DM = matrix(J,M,0);
 for (i = 1; i <= j; i++)
 {
  for (k = 1; k <= m; k++)
  {
   DM(i,k) = StrToInt(StringGrid2->Cells[k][i]);
  }
 }
 DApr = matrix(2,AprS,0);
 for (k = 2; k <= da; k++)
 {
  DApr(1,k) = StrToInt(StringGrid3->Cells[k][1]);
  DApr(2,k) = StrToInt(StringGrid3->Cells[k][2]);
 }
 Opti =  new Optimizavimas(DO, DM, DApr,ProgressBar1);
 DE = Opti->GrazintiDE();
 T = matrix(J,M,0);
 for (int i = 1; i <= J; i++)
 {
  for (int k = 1; k <= M; k++)
  {
   T(i,k) = i;
  }
 }
 ////////////////////////////////
 Vaizd->DuomenysVaizdavimui(J, M, AprS, DO, DM, DE, T, DApr);
 ////////////////////////

 TabSheet2->Enabled = true;
 TabSheet3->Enabled = true;
 TabSheet7->Enabled = true;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
 if( RadioButton1->Checked == true )
  T = Opti->OptimizavimasGenetiskai(StrToInt(Edit2->Text),StrToInt(Edit5->Text),StrToInt(Edit7-
>Text),StrToInt(Edit8->Text),StrToInt(Edit9->Text),StrToInt(Edit10->Text),StrToInt(Edit11->Text));
 if( RadioButton2->Checked == true )
  T = Opti->OptimizavimasGenetiskaiTime(StrToInt(Edit2->Text),StrToInt(Edit5->Text),StrToInt(Edit7-
>Text),StrToInt(Edit8->Text),StrToInt(Edit9->Text),StrToInt(Edit10->Text),StrToInt(Edit11->Text));
 StringGrid4->ColCount = M+1;
 StringGrid4->RowCount = J+1;
 SunumeruotiEilIrStulpE(StringGrid4);
 for (int i = 1; i <= J; i++)
 {
  for (int k = 1; k <= M; k++)
  {
   StringGrid4->Cells[k][i] = IntToStr(T(i,k));
  }
 }
 Label6->Caption = IntToStr(Opti->TvarkarascioIlgis());
```

```
 Vaizd->DuomenysVaizdavimui(T);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button5Click(TObject *Sender)
{
 if( RadioButton1->Checked == true )
  T = Opti->OptimizavimasACO(StrToInt(Edit1->Text),StrToFloat(Edit3->Text),StrToInt(Edit4->Text),
StrToInt(Edit2->Text),StrToInt(Edit5->Text),StrToInt(Edit6->Text),CheckBox1->Checked);
 if( RadioButton2->Checked == true )
  T = Opti->OptimizavimasACOTime(StrToInt(Edit1->Text),StrToFloat(Edit3->Text),StrToInt(Edit4->Text),
StrToInt(Edit2->Text), StrToInt(Edit5->Text), StrToInt(Edit6->Text),CheckBox1->Checked);
 StringGrid4->ColCount = M+1;
 StringGrid4->RowCount = J+1;
 SunumeruotiEilIrStulpE(StringGrid4);
 for (int i = 1; i <= J; i++)
 {
  for (int k = 1; k <= M; k++)
  {
   StringGrid4->Cells[k][i] = IntToStr(T(i,k));
  }
 }
 Label6->Caption = IntToStr(Opti->TvarkarascioIlgis());
 Vaizd->DuomenysVaizdavimui(T);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::PaintBox1Paint(TObject *Sender)
{
 Vaizd->PiestiJ();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::TabSheet3Show(TObject *Sender)
{
 Vaizd->PiestiJ();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::TabSheet7Show(TObject *Sender)
{
 Vaizd->PiestiM();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::PaintBox2Paint(TObject *Sender)
{
 Vaizd->PiestiM();
}
void __fastcall TForm1::Button7Click(TObject *Sender)
{
 Vaizd->GeneruotiNaujasSpalvasJ();
 Vaizd->PiestiJ();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button6Click(TObject *Sender)
{
 Vaizd->GeneruotiNaujasSpalvasM();
 Vaizd->PiestiM();
}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button8Click(TObject *Sender)
{
 StringGrid1->ColCount = 1;
 StringGrid1->RowCount = 1;
 StringGrid2->ColCount = 1;
 StringGrid2->RowCount = 1;
 StringGrid3->ColCount = 1;
 StringGrid3->RowCount = 1;
 Button3->Enabled = false;

}
//---------------------------------------------------------------------------
```

ACO.h

```
//---------------------------------------------------------------------------

#ifndef AcoH
#define AcoH

#include <ComCtrls.hpp>
#include <Math.h>
#include "matrix.h"
#include "matrixr.h"
#include "matrixd.h"
#include "Gerumas.h"
#include "sys\timeb.h"
//---------------------------------------------------------------------------
struct Skruzde
{
 matrix t;   //tvarkarastis
 int ilg;    //jo ilgis
};

class Aco : public Gerumas
{
 private:

    int D; // darbu skaicius
    int M; // masinu (operaciju darbui) skaicius
    matrix DO; // darbas-operacijos laikas
    matrix DM; // darbas-masinos nr
    matrix DE; // darbas-eiliskumas (pagal masinos nr is eiles)
    matrix DApr; // dienu apribojimai
    int AprS; // dienu apribojimu skaicius
    int Skruzd;
    long double Atnauj;
    int AtnaujSk;
    int Iter;
    int Time;
    int Modif;
    bool PapildPedsak;
    matrix T; // Tvarkarastis (sprendinys)
    matrix * Bendr;
    matrixd * Tikimybes;
    Skruzde * Skruzdes;
    int * Rikiuotos;

    TProgressBar *Progresas;

    void SukurtiBendrumoMatricas();
    void SukurtiTikimybiuMatricas();
    void SukurtiSkruzdes();
    void PaleistiSkruzdes();
    void RikiuotiSkruzdesPagalGeruma();
    void AtnaujintiFeromonuPedsakus();
    void Normuoti();
    void Modifikuoti();
    matrix SkruzdeRandom(matrix);

 public:
  Aco(int, int, matrix, matrix, matrix, matrix, int, int, float,int, int,int,int,bool,TProgressBar *);
  matrix Optimizuoti();
  matrix OptimizuotiTime();
};

#endif
```

ACO.cpp

```
//---------------------------------------------------------------------------


#pragma hdrstop

#include "Aco.h"

//---------------------------------------------------------------------------
```

```
#pragma package(smart_init)

//---------------------------------------------------------------------------
Aco::Aco(int d, int m, matrix doo, matrix dm, matrix de, matrix dapr, int aprs, int skruzd,float
anauj,int atnaujSk,int iter,int time,int modif,bool papildpedsak,TProgressBar *progres)
{
 D = d; // darbu skaicius
 M = m; // masinu (operaciju darbui) skaicius
 DO = doo; // darbas-operacijos laikas
 DM = dm; // darbas-masinos nr
 DE = de; // darbas-eiliskumas (pagal masinos nr is eiles)
 DApr = dapr; // dienu apribojimai
 AprS = aprs; // dienu apribojimu skaicius
 Skruzd = skruzd;
 Atnauj = anauj;
 AtnaujSk = atnaujSk;
 Iter = iter;
 Time = time;
 Modif = modif;
 PapildPedsak = papildpedsak;
 Progresas = progres;
// T = matrix(D,M,0);
}
//---------------------------------------------------------------------------
matrix Aco::Optimizuoti()
{
 SukurtiBendrumoMatricas();
 SukurtiTikimybiuMatricas();
 SukurtiSkruzdes();
 Skruzde laik;
 laik.t = matrix(D,M,0);
 laik.ilg = 99999999999;
 Skruzde laik2;
 laik2.t = matrix(D,M,0);
 laik2.ilg = 99999999999;
 int trukm = 99999999999;
 T = matrix(D,M,0);

 for (int i = 1; i <= Iter; i++)
 {
  PaleistiSkruzdes();
  RikiuotiSkruzdesPagalGeruma();
  AtnaujintiFeromonuPedsakus();
  if(trukm > Skruzdes[Rikiuotos[0]-1].ilg)
  {
   trukm = Skruzdes[Rikiuotos[0]-1].ilg;
   T = Skruzdes[Rikiuotos[0]-1].t;
  }

  if(Modif > 0)
   for (int i = 0; i < Modif*2; i++)
   {
    laik.t = SkruzdeRandom(laik.t);
    laik.ilg = GerumoFunkcija(DO, DE, laik.t, DApr);
    if(laik.ilg < laik2.ilg )
    {
     laik2.t = laik.t;
     laik2.ilg = laik.ilg;
    }
   }

  if(Modif > 0)
   if(i%Modif == 0)
    Modifikuoti();

  Progresas->Position = ((float)i/Iter)*100;
 }
 if( GerumoFunkcija(DO, DE, T, DApr) < laik2.ilg ) return T;
 else return laik2.t;
}
//---------------------------------------------------------------------------
matrix Aco::OptimizuotiTime()
{
 SukurtiBendrumoMatricas();
 SukurtiTikimybiuMatricas();
 SukurtiSkruzdes();
 int trukm = 99999999999;
 Skruzde laik;
 laik.t = matrix(D,M,0);
 laik.ilg = 99999999999;
```

```
 Skruzde laik2;
 laik2.t = matrix(D,M,0);
 laik2.ilg = 99999999999;
 T = matrix(D,M,0);
 timeb time1,time2;
 int milis = 0;
 int i = 0;
 ftime(&time1);
 while(milis < Time)
 {
  i++;
  PaleistiSkruzdes();
  RikiuotiSkruzdesPagalGeruma();
  AtnaujintiFeromonuPedsakus();
  if(trukm > Skruzdes[Rikiuotos[0]-1].ilg)
  {
   trukm = Skruzdes[Rikiuotos[0]-1].ilg;
   T = Skruzdes[Rikiuotos[0]-1].t;
  }

  if(Modif > 0)
   for (int i = 0; i < Modif*2; i++)
   {
    laik.t = SkruzdeRandom(laik.t);
    laik.ilg = GerumoFunkcija(DO, DE, laik.t, DApr);
    if(laik.ilg < laik2.ilg )
    {
     laik2.t = laik.t;
     laik2.ilg = laik.ilg;
    }
   }


  if(Modif > 0)
   if(i%Modif == 0)
    Modifikuoti();

  ftime(&time2);
  milis = (time2.time-time1.time)*1000+(time2.millitm-time1.millitm);
  Progresas->Position = ((float)milis/Time)*100;
 }
 if( GerumoFunkcija(DO, DE, T, DApr) < laik2.ilg ) return T;
 else return laik2.t;
}
//---------------------------------------------------------------------------
void Aco::SukurtiBendrumoMatricas()
{
 Bendr = new matrix[M];
 for (int i = 0; i < M; i++)
 {
  Bendr[i] = matrix(D+1,M,0);
 }

 for (int i = 0; i < M; i++)
 {
  for (int k = 1; k <= M; k++)
  {
   for (int l = 1; l <= D; l++)
    if (DM(l,i+1) == k)
    {
     int h = 1;
     while(Bendr[i](h,k)!=0) h++;
     Bendr[i](h,k) = l;
    }
  }
 }

 for (int i = 0; i < M; i++)
 {
  for (int k = 1; k <= M; k++)
  {
   int h = 0;
   for (int l = 1; l <= D; l++)
    if(Bendr[i](l,k)!=0) h++;
   Bendr[i](D+1,k) = h;
  }
 }
}
//---------------------------------------------------------------------------
void Aco::SukurtiTikimybiuMatricas()
```

```
{
 Tikimybes = new matrixd[M];
 for (int i = 0; i < M; i++)
 {
  Tikimybes[i] = matrixd(D,M,0);
 }


 for (int i = 0; i < M; i++)
 {
  for (int k = 1; k <= M; k++)
  {
   for (int l = 1; l <= Bendr[i](D+1,k); l++)
    Tikimybes[i](l,k) = (long double)1/Bendr[i](D+1,k);
  }
 }
}
//---------------------------------------------------------------------------
void Aco::SukurtiSkruzdes()
{
 Skruzdes = new Skruzde[Skruzd];
 for (int i = 0; i < Skruzd; i++)
 {
  Skruzdes[i].t = matrix(D,M,0);
  Skruzdes[i].ilg = 0;
 }
}
//---------------------------------------------------------------------------
void Aco::PaleistiSkruzdes()
{
 long double dalineTik = 1;
 long double atsit = 0;
 long double sumatsit = 0;
 int kuris = 0;
 int kuris2 = 0;
 bool * state = new bool[D];
 for (int s = 0; s < D; s++)
  state[s] = false;
// randomize();
 for (int I = 0; I < Skruzd; I++)
 {
  for (int i = 0; i < M; i++)
  {
   kuris2 = 0;
   for (int k = 1; k <= M; k++)
   {
    dalineTik = 1;
    //dalineTik = 0;

    //for (int s = 1; s <= Bendr[i](D+1,k); s++)
    // dalineTik = dalineTik + Tikimybes[i](s,k);



    for (int s = 0; s < Bendr[i](D+1,k); s++)
     state[s] = false;
    for (int l = 1; l <= Bendr[i](D+1,k); l++)
    {
     kuris2++;
     atsit = dalineTik*((long double)random(100000000))/100000000;
     while(atsit>=sumatsit && kuris<Bendr[i](D+1,k))
     {
      kuris++;
      if(state[kuris-1]==false)
       sumatsit = sumatsit + Tikimybes[i](kuris,k);
     }
     state[kuris-1]=true;
     dalineTik = dalineTik - Tikimybes[i](kuris,k);
     Skruzdes[I].t(Bendr[i](kuris,k),i+1) = kuris2;
     sumatsit = 0;
     kuris = 0;
    }
   }
  }
 }
 for (int i = 0; i < Skruzd; i++)
  Skruzdes[i].ilg = GerumoFunkcija(DO, DE, Skruzdes[i].t, DApr);
}
//---------------------------------------------------------------------------
void Aco::RikiuotiSkruzdesPagalGeruma()
```

```
{
 Rikiuotos = new int[Skruzd];
 int laik;
 for (int i = 0; i < Skruzd; i++)
  Rikiuotos[i]=i+1;
 for (int i = 0; i < Skruzd-1; i++)
  for (int k = i+1; k < Skruzd; k++)
   if(Skruzdes[Rikiuotos[i]-1].ilg>Skruzdes[Rikiuotos[k]-1].ilg)
   {
    laik = Rikiuotos[k];
    Rikiuotos[k] = Rikiuotos[i];
    Rikiuotos[i] = laik;
   }
}
//---------------------------------------------------------------------------
void Aco::AtnaujintiFeromonuPedsakus()
{
 int kuris = 0;
 int kuris2 = 0;
 if(PapildPedsak == false)
 {
  for (int I = 0; I < AtnaujSk; I++)
  {
   for (int i = 0; i < M; i++)
   {
    kuris2 = 0;
    for (int k = 1; k <= M; k++)
    {
     for (int l = 1; l <= Bendr[i](D+1,k); l++)
     {
      kuris++;
      if(Skruzdes[Rikiuotos[I]-1].t(Bendr[i](kuris,k),i+1) == kuris2+1)
       Tikimybes[i](kuris,k) = Tikimybes[i](kuris,k) + Tikimybes[i](kuris,k)*Atnauj/AtnaujSk;
     }
     kuris2 = kuris2 + kuris;
     kuris = 0;
    }
   }
  }
 }
 else
 {
  for (int I = 0; I < AtnaujSk; I++)
  {
   for (int i = 0; i < M; i++)
   {
    kuris2 = 0;
    for (int k = 1; k <= M; k++)
    {
     for (int l = 1; l <= Bendr[i](D+1,k); l++)
     {
      kuris++;
      for (int x = 1; x <= Bendr[i](D+1,k)-1; x++)
       if(Skruzdes[Rikiuotos[I]-1].t(Bendr[i](kuris,k),i+1) == kuris2+x)
        Tikimybes[i](kuris,k) = Tikimybes[i](kuris,k) + Tikimybes[i](kuris,k)*Atnauj/AtnaujSk *(1 -
((long double)(x-1)/(Bendr[i](D+1,k)-1)));
     }
     kuris2 = kuris2 + kuris;
     kuris = 0;
    }
   }
  }
 }
 Normuoti();
}
//---------------------------------------------------------------------------
void Aco::Normuoti()
{
 long double laik = 0;
 for (int i = 0; i < M; i++)
 {
  for (int k = 1; k <= M; k++)
  {
   for (int l = 1; l <= Bendr[i](D+1,k); l++)
   {
    if(Tikimybes[i](l,k) < 0.0000000000001)
     Tikimybes[i](l,k) = 0.0000000000001;
    laik = laik + Tikimybes[i](l,k);
   }
   for (int l = 1; l <= Bendr[i](D+1,k); l++)
```

```
    Tikimybes[i](l,k) = Tikimybes[i](l,k)/laik;
   laik = 0;
  }
 }
}
//---------------------------------------------------------------------------
void Aco::Modifikuoti()
{
 int i = random(M);

 for (int k = 1; k <= M; k++)
  for (int l = 1; l <= Bendr[i](D+1,k); l++)
   Tikimybes[i](l,k) = 1/Tikimybes[i](l,k);       //((long double)random(100000000))/100000000;;

 Normuoti();
}
//---------------------------------------------------------------------------
matrix Aco::SkruzdeRandom(matrix X)
{
 int jj = X.nrows();
 int mm = X.ncols();
 int * RndMas = new int[jj];
 int * RndMasNr = new int[jj];

 //randomize();

 for (int i = 1; i <= mm; i++)
 {

  for (int h = 0; h < jj; h++)
  {
   RndMas[h] = random(100000000);
   RndMasNr[h] = 0;
  }

  for (int h = 0; h < jj; h++)
  {
   for (int j = 0; j < jj; j++)
   {
    if(h!=j)
    {
     if(RndMas[h]<=RndMas[j])
      RndMasNr[h]++;
     if(RndMas[h]==RndMas[j] && h<j)
      RndMasNr[h]--;
    }
   }
  }


  for (int k = 1; k <= jj; k++)     //atsitiktine seka priskiriama vienam stulpeliui, kitaip tariant
suformuojamas atsitiktinis genas
  {
   X(k,i) = RndMasNr[k-1]+1;
   //RndMas[k-1] = 0;
  }

 }

 return X;
}
//---------------------------------------------------------------------------
```

Genetinis.h

```
//---------------------------------------------------------------------------

#ifndef GenetinisH
#define GenetinisH

#include <ComCtrls.hpp>
#include <Math.h>
#include "matrix.h"
#include "matrixr.h"
#include "matrixd.h"
#include "Gerumas.h"
#include "sys\timeb.h"
//---------------------------------------------------------------------------
```

```
struct Individas
{
 matrix t;   //tvarkarastis
 int ilg;    //jo ilgis
};

class Genetinis : public Gerumas
{
 private:
    int D; // darbu skaicius
    int M; // masinu (operaciju darbui) skaicius
    matrix DO; // darbas-operacijos laikas
    matrix DM; // darbas-masinos nr
    matrix DE; // darbas-eiliskumas (pagal masinos nr is eiles)
    matrix DApr; // dienu apribojimai
    int AprS; // dienu apribojimu skaicius
    int Iter;
    int Time;
    int Individ;
    int Elite;
    int Pakeiciami;
    int MutIndiv;
    int MutGen;

    matrix T; // Tvarkarastis (sprendinys)
    matrix * Bendr;
    matrixd * Tikimybes;
    Individas * Individai;
    int * Rikiuoti;


    TProgressBar *Progresas;


    void SukurtiBendrumoMatricas();
    void SukurtiTikimybiuMatricas();
    void SukurtiIndividus();
    matrix IndividasRandom(matrix);
    void GeneruotiPradinePopuliacija();
    bool tikrinti(matrix);
    void RikiuotiIndividusPagalGeruma();
    void PakeistiPrasciausius();
    void Kryzminti();
    void KryzmintiIndividus(int i, int a);
    void PerskaiciuotiLaikus();
    void Mutuoti();

 public:
  Genetinis(int, int, matrix, matrix, matrix, matrix, int, int, int, int, int, int, int, int,
TProgressBar *);
  matrix Optimizuoti();
  matrix OptimizuotiTime();
};
#endif
```

Genetinis.cpp

```
//---------------------------------------------------------------------------


#pragma hdrstop

#include "Genetinis.h"

//---------------------------------------------------------------------------

#pragma package(smart_init)

//---------------------------------------------------------------------------
Genetinis::Genetinis(int d, int m, matrix doo, matrix dm, matrix de, matrix dapr, int aprs,
 int iter,
 int time,
 int individ,
 int elite,
 int pakeiciami,
 int mutindiv,
 int mutgen,
 TProgressBar *progres)
```

```
{
 D = d; // darbu skaicius
 M = m; // masinu (operaciju darbui) skaicius
 DO = doo; // darbas-operacijos laikas
 DM = dm; // darbas-masinos nr
 DE = de; // darbas-eiliskumas (pagal masinos nr is eiles)
 DApr = dapr; // dienu apribojimai
 AprS = aprs; // dienu apribojimu skaicius
 Iter = iter;
 Time = time;
 Individ = individ;
 Elite = elite;
 Pakeiciami = pakeiciami;
 MutIndiv = mutindiv;
 MutGen = mutgen;

 Progresas = progres;
                                        // T = matrix(D,M,0);
}
//---------------------------------------------------------------------------
matrix Genetinis::Optimizuoti()
{
 //SukurtiBendrumoMatricas();
 //SukurtiTikimybiuMatricas();
 SukurtiIndividus();
 int trukm = 99999999999;
 T = matrix(D,M,0);
 GeneruotiPradinePopuliacija();
 RikiuotiIndividusPagalGeruma();
 T = Individai[Rikiuoti[0]-1].t;
 trukm = Individai[Rikiuoti[0]-1].ilg;

 /*for (int i = 1; i <= D; i++)
 {
  for (int k = 1; k <= M; k++)
  {
   T(i,k) = i;
  }
 }   */



 for (int i = 1; i <= Iter; i++)
 {
  PakeistiPrasciausius();
  Kryzminti();
  Mutuoti();
  PerskaiciuotiLaikus();


  RikiuotiIndividusPagalGeruma();
  if(trukm > Individai[Rikiuoti[0]-1].ilg)
  {
   trukm = Individai[Rikiuoti[0]-1].ilg;
   T = Individai[Rikiuoti[0]-1].t;
  }

  Progresas->Position = ((float)i/Iter)*100;
 }

 return T;
}
//---------------------------------------------------------------------------
matrix Genetinis::OptimizuotiTime()
{
// SukurtiBendrumoMatricas();
// SukurtiTikimybiuMatricas();
// SukurtiSkruzdes();
 SukurtiIndividus();
 int trukm = 99999999999;
 T = matrix(D,M,0);
 GeneruotiPradinePopuliacija();
 RikiuotiIndividusPagalGeruma();
 T = Individai[Rikiuoti[0]-1].t;
 trukm = Individai[Rikiuoti[0]-1].ilg;
 timeb time1,time2;
 int milis = 0;
 int i = 0;
 ftime(&time1);
 while(milis < Time)
```

```
 {
  PakeistiPrasciausius();
  Kryzminti();
  Mutuoti();
  PerskaiciuotiLaikus();


  RikiuotiIndividusPagalGeruma();
  if(trukm > Individai[Rikiuoti[0]-1].ilg)
  {
   trukm = Individai[Rikiuoti[0]-1].ilg;
   T = Individai[Rikiuoti[0]-1].t;
  }

  ftime(&time2);
  milis = (time2.time-time1.time)*1000+(time2.millitm-time1.millitm);
  Progresas->Position = ((float)milis/Time)*100;
 }
 return T;
}
//---------------------------------------------------------------------------
void Genetinis::SukurtiBendrumoMatricas()
{
 Bendr = new matrix[M];
 for (int i = 0; i < M; i++)
 {
  Bendr[i] = matrix(D+1,M,0);
 }

 for (int i = 0; i < M; i++)
 {
  for (int k = 1; k <= M; k++)
  {
   for (int l = 1; l <= D; l++)
    if (DM(l,i+1) == k)
    {
     int h = 1;
     while(Bendr[i](h,k)!=0) h++;
     Bendr[i](h,k) = l;
    }
  }
 }

 for (int i = 0; i < M; i++)
 {
  for (int k = 1; k <= M; k++)
  {
   int h = 0;
   for (int l = 1; l <= D; l++)
    if(Bendr[i](l,k)!=0) h++;
   Bendr[i](D+1,k) = h;
  }
 }
}
//---------------------------------------------------------------------------
void Genetinis::SukurtiIndividus()
{
 Individai = new Individas[Individ];
 for (int i = 0; i < Individ; i++)
 {
  Individai[i].t = matrix(D,M,0);
  Individai[i].ilg = 0;
 }
}
//---------------------------------------------------------------------------
void Genetinis::GeneruotiPradinePopuliacija()
{

 for (int i = 0; i < Individ; i++)
 {
  Individai[i].t = IndividasRandom(Individai[i].t);
  Individai[i].ilg = GerumoFunkcija(DO, DE, Individai[i].t, DApr);
 }

}
//---------------------------------------------------------------------------
matrix Genetinis::IndividasRandom(matrix X)
{
 int jj = X.nrows();
 int mm = X.ncols();
```

```
 int * RndMas = new int[jj];
 int * RndMasNr = new int[jj];

 for (int i = 1; i <= mm; i++)
 {

  for (int h = 0; h < jj; h++)
  {
   RndMas[h] = random(100000000);
   RndMasNr[h] = 0;
  }

  for (int h = 0; h < jj; h++)
  {
   for (int j = 0; j < jj; j++)
   {
    if(h!=j)
    {
     if(RndMas[h]<=RndMas[j])
      RndMasNr[h]++;
     if(RndMas[h]==RndMas[j] && h<j)
      RndMasNr[h]--;
    }
   }
  }


  for (int k = 1; k <= jj; k++)     //atsitiktine seka priskiriama vienam stulpeliui, kitaip tariant
suformuojamas atsitiktinis genas
  {
   X(k,i) = RndMasNr[k-1]+1;
   //RndMas[k-1] = 0;
  }

 }

 if(!tikrinti(X))Application->MessageBox("crap", "Warning", MB_OK);
 return X;

}
//---------------------------------------------------------------------------
bool Genetinis::tikrinti(matrix X)
{
 for (int i = 1; i <= M; i++)
 {
  for (int h = 1; h <= D; h++)
  {
   for (int l = 1; l <= D; l++)
   {
    if (h!=l)
    {
     if (X(h,i)==X(l,i) || X(h,i) <= 0 || X(l,i) <= 0 || X(h,i) > 20 || X(l,i) > 20) return false;
    }
   }
  }
 }
 return true;
}
//---------------------------------------------------------------------------
void Genetinis::RikiuotiIndividusPagalGeruma()
{
 Rikiuoti = new int[Individ];
 int laik;
 for (int i = 0; i < Individ; i++)
  Rikiuoti[i]=i+1;
 for (int i = 0; i < Individ-1; i++)
  for (int k = i+1; k < Individ; k++)
   if(Individai[Rikiuoti[i]-1].ilg>Individai[Rikiuoti[k]-1].ilg)
   {
    laik = Rikiuoti[k];
    Rikiuoti[k] = Rikiuoti[i];
    Rikiuoti[i] = laik;
   }
}
//---------------------------------------------------------------------------
void Genetinis::SukurtiTikimybiuMatricas()
{
 Tikimybes = new matrixd[M];
 for (int i = 0; i < M; i++)
 {
```

```
  Tikimybes[i] = matrixd(D,M,0);
 }


 for (int i = 0; i < M; i++)
 {
  for (int k = 1; k <= M; k++)
  {
   for (int l = 1; l <= Bendr[i](D+1,k); l++)
    Tikimybes[i](l,k) = (long double)1/Bendr[i](D+1,k);
  }
 }
}
//---------------------------------------------------------------------------
void Genetinis::PakeistiPrasciausius()
{

 for (int i = Individ - Pakeiciami; i < Individ; i++)
 {
  Individai[i].t = IndividasRandom(Individai[i].t);
  //Individai[i].ilg = GerumoFunkcija(DO, DE, Individai[i].t, DApr);
 }

}
//---------------------------------------------------------------------------
void Genetinis::Kryzminti()
{
 int a, b;
 for (int i = Elite ; i < Individ - Pakeiciami; i++)
 {
  a = random(Individ);
  KryzmintiIndividus(i,a);
 }
}
//---------------------------------------------------------------------------
void Genetinis::KryzmintiIndividus(int i, int a)
{
 int kurisPirmas = random(1);
 int kur = random(M-1) + 2;
 matrix LaikT = matrix(D,M,0);
 if(kurisPirmas == 0)
 {
  for (int j = 1; j < kur; j++)
  {
   for (int h = 1; h <= D; h++)
   {
    LaikT(h,j)= Individai[i].t(h,j);
   }
  }
  for (int j = kur; j <= M; j++)
  {
   for (int h = 1; h <= D; h++)
   {
    LaikT(h,j)= Individai[a].t(h,j);
   }
  }
 }
 else
 {
  for (int j = 1; j < kur; j++)
  {
   for (int h = 1; h <= D; h++)
   {
    LaikT(h,j)= Individai[a].t(h,j);
   }
  }
  for (int j = kur; j <= M; j++)
  {
   for (int h = 1; h <= D; h++)
   {
    LaikT(h,j)= Individai[i].t(h,j);
   }
  }
 }
 Individai[i].t = LaikT;
}
//---------------------------------------------------------------------------
void Genetinis::PerskaiciuotiLaikus()
{
 for (int i = 0; i < Individ; i++)
```

```
 {
  Individai[i].ilg = GerumoFunkcija(DO, DE, Individai[i].t, DApr);
 }
}
//---------------------------------------------------------------------------
void Genetinis::Mutuoti()
{
 int a = 0;
 int b = 0;
 int b1 = 0;
 int b2 = 0;
 int laik = 0;
 for (int x = 0; x < MutIndiv; x++)
 {
  a = random(Individ-Elite-Pakeiciami)+Elite;
  for (int y = 0; y < MutGen; y++)
  {
   b = random(M)+1;
   b1 = random(D)+1;
   b2 = random(D)+1;
   laik = Individai[a].t(b1,b);
   Individai[a].t(b1,b) = Individai[a].t(b2,b);
   Individai[a].t(b2,b) = laik;
  }
 }
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
```

Gerumas.h

```
//---------------------------------------------------------------------------

#ifndef GerumasH
#define GerumasH

#include "matrix.h"
//---------------------------------------------------------------------------
class Gerumas
{
 private:

 public:
  int GerumoFunkcija(matrix, matrix, matrix, matrix);
  matrix Vaizdas(matrix, matrix, matrix, matrix);
  int GerumoFunkcijaBeApr(matrix, matrix, matrix);
  matrix VaizdasBeApr(matrix, matrix, matrix);
};
#endif
```

Gerumas.cpp

```
//---------------------------------------------------------------------------


#pragma hdrstop

#include "Gerumas.h"

//---------------------------------------------------------------------------

#pragma package(smart_init)

//---------------------------------------------------------------------------
int Gerumas::GerumoFunkcija(matrix o, matrix e, matrix t, matrix ap)
{
 int d = o.nrows();
 int m = o.ncols();
 int app = ap.ncols();
 matrix V = matrix(d,m,0);
 matrix te = matrix(d,m,0);    //perrikiuotas tvarkarastis pagal eiliskuma
 for (int k = 1; k <= m; k++)
 {
  for (int i = 1; i <= d; i++)
```

```
  {
   te(t(i,k),k) = i;
  }
 }

 if(app == 1)
 {
  int Laik = 0;

  for (int i = 1; i <= d; i++)
  {
   for (int ii = 1; ii < i; ii++)
    if (e(te(ii,1),1) == e(te(i,1),1) && Laik < V(te(ii,1),1)) Laik = V(te(ii,1),1);
   V(te(i,1),1) = Laik + o(te(i,1),e(te(i,1),1));
   Laik = 0;
  }

  for (int k = 2; k <= m; k++)
  {
   for (int i = 1; i <= d; i++)
   {
    for (int ii = 1; ii <= d; ii++)
     if (e(ii,k-1) == e(te(i,k),k) && Laik < V(ii,k-1)) Laik = V(ii,k-1);
    for (int ii = 1; ii < i; ii++)
     if (e(te(ii,k),k) == e(te(i,k),k) && Laik < V(te(ii,k),k)) Laik = V(te(ii,k),k);
    if(Laik > V(te(i,k),k-1)) V(te(i,k),k) = Laik + o(te(i,k),e(te(i,k),k));
    else V(te(i,k),k) = V(te(i,k),k-1) + o(te(i,k),e(te(i,k),k));
    Laik = 0;
   }
  }

  for (int i = 1; i <= d; i++)
   if(Laik < V(i,m)) Laik = V(i,m);
  return Laik;
 }

 else
 {
  int Laik = 0;

  for (int i = 1; i <= d; i++)
  {
   for (int ii = 1; ii < i; ii++)
    if (e(te(ii,1),1) == e(te(i,1),1) && Laik < V(te(ii,1),1)) Laik = V(te(ii,1),1);
   V(te(i,1),1) = Laik + o(te(i,1),e(te(i,1),1));

   for (int zz = 2; zz <= app; zz++)
    if( ap(1,zz) < V(te(i,1),1) )
     V(te(i,1),1) = ap(2,zz) + o(te(i,1),e(te(i,1),1));

   Laik = 0;
  }

  for (int k = 2; k <= m; k++)
  {
   for (int i = 1; i <= d; i++)
   {
    for (int ii = 1; ii <= d; ii++)
     if (e(ii,k-1) == e(te(i,k),k) && Laik < V(ii,k-1)) Laik = V(ii,k-1);
    for (int ii = 1; ii < i; ii++)
     if (e(te(ii,k),k) == e(te(i,k),k) && Laik < V(te(ii,k),k)) Laik = V(te(ii,k),k);
    if(Laik > V(te(i,k),k-1)) V(te(i,k),k) = Laik + o(te(i,k),e(te(i,k),k));
    else V(te(i,k),k) = V(te(i,k),k-1) + o(te(i,k),e(te(i,k),k));

    for (int zz = 2; zz <= app; zz++)
     if( (ap(1,zz) < V(te(i,k),k)) && (ap(1,zz) >= V(te(i,k),k)-o(te(i,k),e(te(i,k),k))) )
      V(te(i,k),k) = ap(2,zz) + o(te(i,k),e(te(i,k),k));

    Laik = 0;
   }
  }

  for (int i = 1; i <= d; i++)
   if(Laik < V(i,m)) Laik = V(i,m);
  return Laik;
 }
}
//---------------------------------------------------------------------------
matrix Gerumas::Vaizdas(matrix o, matrix e, matrix t, matrix ap)
{
```

```
int d = o.nrows();
int m = o.ncols();
int app = ap.ncols();
matrix V = matrix(d,m,0);
matrix te = matrix(d,m,0);    //perrikiuotas tvarkarastis pagal eiliskuma
for (int k = 1; k <= m; k++)
{
 for (int i = 1; i <= d; i++)
 {
  te(t(i,k),k) = i;
 }
}

if(app == 1)
{
 int Laik = 0;

 for (int i = 1; i <= d; i++)
 {
  for (int ii = 1; ii < i; ii++)
   if (e(te(ii,1),1) == e(te(i,1),1) && Laik < V(te(ii,1),1)) Laik = V(te(ii,1),1);
  V(te(i,1),1) = Laik + o(te(i,1),e(te(i,1),1));
  Laik = 0;
 }

 for (int k = 2; k <= m; k++)
 {
  for (int i = 1; i <= d; i++)
  {
   for (int ii = 1; ii <= d; ii++)
    if (e(ii,k-1) == e(te(i,k),k) && Laik < V(ii,k-1)) Laik = V(ii,k-1);
   for (int ii = 1; ii < i; ii++)
    if (e(te(ii,k),k) == e(te(i,k),k) && Laik < V(te(ii,k),k)) Laik = V(te(ii,k),k);
   if(Laik > V(te(i,k),k-1)) V(te(i,k),k) = Laik + o(te(i,k),e(te(i,k),k));
   else V(te(i,k),k) = V(te(i,k),k-1) + o(te(i,k),e(te(i,k),k));
   Laik = 0;
  }
 }

 return V;
}

else
{
 int Laik = 0;

 for (int i = 1; i <= d; i++)
 {
  for (int ii = 1; ii < i; ii++)
   if (e(te(ii,1),1) == e(te(i,1),1) && Laik < V(te(ii,1),1)) Laik = V(te(ii,1),1);
  V(te(i,1),1) = Laik + o(te(i,1),e(te(i,1),1));

  for (int zz = 2; zz <= app; zz++)
   if( ap(1,zz) < V(te(i,1),1) )
    V(te(i,1),1) = ap(2,zz) + o(te(i,1),e(te(i,1),1));

  Laik = 0;
 }

 for (int k = 2; k <= m; k++)
 {
  for (int i = 1; i <= d; i++)
  {
   for (int ii = 1; ii <= d; ii++)
    if (e(ii,k-1) == e(te(i,k),k) && Laik < V(ii,k-1)) Laik = V(ii,k-1);
   for (int ii = 1; ii < i; ii++)
    if (e(te(ii,k),k) == e(te(i,k),k) && Laik < V(te(ii,k),k)) Laik = V(te(ii,k),k);
   if(Laik > V(te(i,k),k-1)) V(te(i,k),k) = Laik + o(te(i,k),e(te(i,k),k));
   else V(te(i,k),k) = V(te(i,k),k-1) + o(te(i,k),e(te(i,k),k));

   for (int zz = 2; zz <= app; zz++)
    if( (ap(1,zz) < V(te(i,k),k)) && (ap(1,zz) >= V(te(i,k),k)-o(te(i,k),e(te(i,k),k))) )
     V(te(i,k),k) = ap(2,zz) + o(te(i,k),e(te(i,k),k));

   Laik = 0;
  }
 }

 return V;
}
```

```
}
//---------------------------------------------------------------------------
int Gerumas::GerumoFunkcijaBeApr(matrix o, matrix e, matrix t)
{
 int d = o.nrows();
 int m = o.ncols();
 matrix V = matrix(d,m,0);
 matrix te = matrix(d,m,0);    //perrikiuotas tvarkarastis pagal eiliskuma
 for (int k = 1; k <= m; k++)
 {
  for (int i = 1; i <= d; i++)
  {
   te(t(i,k),k) = i;
  }
 }

 int Laik = 0;

 for (int i = 1; i <= d; i++)
 {
  for (int ii = 1; ii < i; ii++)
   if (e(te(ii,1),1) == e(te(i,1),1) && Laik < V(te(ii,1),1)) Laik = V(te(ii,1),1);
 V(te(i,1),1) = Laik + o(te(i,1),e(te(i,1),1));
 Laik = 0;
 }

 for (int k = 2; k <= m; k++)
 {
  for (int i = 1; i <= d; i++)
  {
   for (int ii = 1; ii <= d; ii++)
    if (e(ii,k-1) == e(te(i,k),k) && Laik < V(ii,k-1)) Laik = V(ii,k-1);
   for (int ii = 1; ii < i; ii++)
    if (e(te(ii,k),k) == e(te(i,k),k) && Laik < V(te(ii,k),k)) Laik = V(te(ii,k),k);
   if(Laik > V(te(i,k),k-1)) V(te(i,k),k) = Laik + o(te(i,k),e(te(i,k),k));
   else V(te(i,k),k) = V(te(i,k),k-1) + o(te(i,k),e(te(i,k),k));
   Laik = 0;
  }
 }

 for (int i = 1; i <= d; i++)
  if(Laik < V(i,m)) Laik = V(i,m);
 return Laik;
}
//---------------------------------------------------------------------------
matrix Gerumas::VaizdasBeApr(matrix o, matrix e, matrix t)
{
 int d = o.nrows();
 int m = o.ncols();
 matrix V = matrix(d,m,0);
 matrix te = matrix(d,m,0);    //perrikiuotas tvarkarastis pagal eiliskuma
 for (int k = 1; k <= m; k++)
 {
  for (int i = 1; i <= d; i++)
  {
   te(t(i,k),k) = i;
  }
 }

 int Laik = 0;

 for (int i = 1; i <= d; i++)
 {
  for (int ii = 1; ii < i; ii++)
   if (e(te(ii,1),1) == e(te(i,1),1) && Laik < V(te(ii,1),1)) Laik = V(te(ii,1),1);
 V(te(i,1),1) = Laik + o(te(i,1),e(te(i,1),1));
 Laik = 0;
 }

 for (int k = 2; k <= m; k++)
 {
  for (int i = 1; i <= d; i++)
  {
   for (int ii = 1; ii <= d; ii++)
    if (e(ii,k-1) == e(te(i,k),k) && Laik < V(ii,k-1)) Laik = V(ii,k-1);
   for (int ii = 1; ii < i; ii++)
    if (e(te(ii,k),k) == e(te(i,k),k) && Laik < V(te(ii,k),k)) Laik = V(te(ii,k),k);
   if(Laik > V(te(i,k),k-1)) V(te(i,k),k) = Laik + o(te(i,k),e(te(i,k),k));
   else V(te(i,k),k) = V(te(i,k),k-1) + o(te(i,k),e(te(i,k),k));
   Laik = 0;
```

```
  }
 }

 return V;
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
```

Matrix.h

```
//---------------------------------------------------------------------------

#ifndef matrixH
#define matrixH
//---------------------------------------------------------------------------
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h> // needed for the exit function

class matrix
{
   private:
      int rows;          // number of rows
      int cols;          // number of columns
      int ** element;  // pointer to the row pointers

      // These utilities are need by the other members
      void setup(int m, int n); // sets up the matrix in memory
      void destroy(void);       // releases the memory

   public:
      matrix(int m=1, int n=1, int s=0);  // constructor of an m by n matrix
                                          // filled with the constant s
      matrix(const matrix& x);            // copy constructor
      ~matrix(void);                      // destructor
      matrix& operator = (const matrix& x);   // operator =
      int nrows(void);                    // returns rows
      int ncols(void);                    // returns cols
      int& operator () (int i, int j);      // returns element[i][j];

   friend istream& operator >> (istream& in, matrix& x);
};



#endif
```

Matrix.cpp

```
//---------------------------------------------------------------------------


#pragma hdrstop

#include "matrix.h"

//---------------------------------------------------------------------------

#pragma package(smart_init)

void matrix::setup(int m, int n)
{
   int i;

   // allocate the array of pointers to the rows of the matrix
   element = new int* [m];

   // offset by 1
   element = element - 1;

   // now allocate enough space to hold the entries
   element[1] = new int[m*n];

   // offset the first row by 1
   element[1] = element[1] - 1;
```

```
   // now do the rest
   for(i=2; i<=m; i++)
      element[i] = element[i-1] +n;
}

void matrix::destroy(void)
{
   // release the memory in the reverse order in which it was allocated
   delete [] (element[1] + 1);
   delete [] (element + 1);
}

matrix::matrix(int m, int n, int s)
{
   int i, j;
   rows = m;
   cols = n;
   setup(m,n);
   for(i=1; i<=m; i++) for(j=1; j<=n; j++) element[i][j] = s;
}

matrix::matrix(const matrix& x)
{
   int i, j;

   rows = x.rows;
   cols = x.cols;
   setup(rows,cols);

   for(i=1; i<=rows; i++)
      for(j=1; j<=cols; j++)
         element[i][j] = x.element[i][j];
}

matrix& matrix::operator = (const matrix& x)
{
   int i, j;

   if (element == x.element) // nothing to do
      return *this;

   if (!((rows == x.rows)&&(cols == x.cols)) ) // must call reallocate
   {
      destroy();
      rows = x.rows;
      cols = x.cols;
      setup(rows,cols);
   }

   for(i=1; i<=rows; i++)
      for(j=1; j<=cols; j++)
         element[i][j] = x.element[i][j];

   return *this;
}

matrix::~matrix(void)
{
   destroy();
}

int matrix::nrows(void)
{
   return rows;
}

int matrix::ncols(void)
{
   return cols;
}

int& matrix::operator () (int i, int j)
{
   return element[i][j];
}

/////////////////  matrix operations ////////////////////////////////

istream& operator >> (istream &in, matrix &x)
{
```

```
   int m, n, i, j;

   in >> m >> n;

   // check whether the sizes are compatible
   if(!( (m == x.rows) && (n == x.cols) ) )
   {
      x.destroy();
      x.setup(m,n);
      x.rows = m;
      x.cols = n;
   }

   for(i=1; i <= m; i++)
      for(j=1; j <= n; j++)
         in >> x(i,j);

   return in;
}

ostream& operator << (ostream &out, matrix &x)
{
   int i, j;
   int m = x.nrows(), n = x.ncols();

   for(i=1; i <= m; i++)
   {
      for(j=1; j<=n; j++)
         out << x(i,j)<< '\t';

      out << endl;
   }

   return out;
}

matrix operator + (matrix &x, matrix &y)
{
   int i, j;
   int m = x.nrows(), n = x.ncols();

   if(! ( (x.nrows() == y.nrows()) && (x.ncols() == y.ncols() ) ) )
   {
      cout << "matrices incompatible\n";
      exit(1);
   }

   matrix c(m,n);

   for(i=1; i <= m; i++)
      for(j=1; j <= n; j++)
         c(i,j) = x(i,j) + y(i,j);

   return c;
}

matrix operator * (matrix &x, matrix &y)
{
   int i, j, k;
   int m = x.nrows(), n = x.ncols(), p = y.ncols();
   int sum;

   if( y.nrows() != n )
   {
      cout << "matrices incompatible\n";
      exit(1);
   }

   matrix c(m,p);

   for(i=1; i <= m; i++)
      for(j=1; j <= p; j++)
      {
         sum = 0.0f;
         for(k=1; k<=n; k++)
            sum = sum + x(i,k)*y(k,j);

         c(i,j) = sum;
      }
```

```
   return c;
}
```

Optimizavimas.h

```
//---------------------------------------------------------------------------

#ifndef OptimizavimasH
#define OptimizavimasH


#include <ComCtrls.hpp>
#include "matrix.h"
#include "Gerumas.h"
#include "Aco.h"
#include "Genetinis.h"
//---------------------------------------------------------------------------
class Optimizavimas : public Gerumas
{
   private:
    int D; // darbu skaicius
    int M; // masinu (operaciju darbui) skaicius
    matrix DO; // darbas-operacijos laikas
    matrix DM; // darbas-masinos nr
    matrix DE; // darbas-eiliskumas (pagal masinos nr is eiles)
    matrix DApr; // dienu apribojimai
    int AprS; // dienu apribojimu skaicius
    matrix T; // Tvarkarastis (sprendinys)
    TProgressBar * Progresas;

   public:
    Optimizavimas(){};
    Optimizavimas(matrix, matrix, matrix,TProgressBar *);

    matrix OptimizavimasGenetiskai(int,int,int,int,int,int,int);
    matrix OptimizavimasGenetiskaiTime(int,int,int,int,int,int,int);
    matrix OptimizavimasACO(int,float,int,int,int,int,bool);
    matrix OptimizavimasACOTime(int,float,int,int,int,int,bool);
    matrix GrazintiDE();
    int TvarkarascioIlgis();
    int TvarkarascioIlgis(matrix, matrix, matrix, matrix);

};


#endif
```

Optimizavimas.cpp

```
//---------------------------------------------------------------------------


#pragma hdrstop

#include "Optimizavimas.h"

//---------------------------------------------------------------------------

#pragma package(smart_init)
//---------------------------------------------------------------------------

Optimizavimas::Optimizavimas(matrix doo, matrix dmm, matrix dap, TProgressBar *progres)
{
 DO = doo;
 DM = dmm;
 DApr = dap;
 D = DM.nrows();
 M = DM.ncols();
 AprS = DApr.ncols();
 DE = matrix(D,M,0);
 Progresas = progres;
 for (int i = 1; i <= D; i++)
 {
  for (int k = 1; k <= M; k++)
  {
   DE(i,DM(i,k)) = k;
```

```
  }
 }
 T = matrix(1,1,1);
}
//---------------------------------------------------------------------------
matrix Optimizavimas::OptimizavimasGenetiskai(int iter, int time, int individ, int elite, int
pakeiciami, int mutindiv, int mutgen)
{
 T = matrix(D,M,0);
 Genetinis * genetinis = new Genetinis(D, M, DO, DM, DE, DApr, AprS, iter, time, individ, elite,
pakeiciami, mutindiv, mutgen, Progresas);
 T = genetinis->Optimizuoti();
 delete genetinis;
 return T;

 /*

 T = matrix(D,M,0);
 for (int i = 1; i <= D; i++)
 {
  for (int k = 1; k <= M; k++)
  {
   T(i,k) = i;
  }
 }
 return T;   */
}
//---------------------------------------------------------------------------
matrix Optimizavimas::OptimizavimasGenetiskaiTime(int iter, int time, int individ, int elite, int
pakeiciami, int mutindiv, int mutgen)
{
 T = matrix(D,M,0);
 Genetinis * genetinis = new Genetinis(D, M, DO, DM, DE, DApr, AprS, iter, time, individ, elite,
pakeiciami, mutindiv, mutgen, Progresas);
 T = genetinis->OptimizuotiTime();
 delete genetinis;
 return T;
}
//---------------------------------------------------------------------------
matrix Optimizavimas::OptimizavimasACO(int skruzd,float anauj,int anaujSk,int iter,int time,int
modif,bool papildpedsak)
{
 T = matrix(D,M,0);
 Aco * aco = new Aco(D, M, DO, DM, DE, DApr, AprS, skruzd, anauj, anaujSk, iter, time,
modif,papildpedsak, Progresas);
  T = aco->Optimizuoti();
 delete aco;


/* for (int i = 1; i <= D; i++)
 {
  for (int k = 1; k <= M; k++)
  {
   T(i,k) = D - i + 1;
  }
 }   */
/* T = matrix(3,3,0);
 T(1,1) = 3;
 T(2,1) = 1;
 T(3,1) = 2;
 T(1,2) = 2;
 T(2,2) = 3;
 T(3,2) = 1;
 T(1,3) = 1;
 T(2,3) = 3;
 T(3,3) = 2;   */
 return T;
}
//---------------------------------------------------------------------------
matrix Optimizavimas::OptimizavimasACOTime(int skruzd,float anauj,int anaujSk,int iter,int time,int
modif,bool papildpedsak)
{
 T = matrix(D,M,0);
 Aco * aco = new Aco(D, M, DO, DM, DE, DApr, AprS, skruzd, anauj, anaujSk, iter,
time,modif,papildpedsak, Progresas);
 T = aco->OptimizuotiTime();
 delete aco;
 return T;
}
//---------------------------------------------------------------------------
```

```
int Optimizavimas::TvarkarascioIlgis(matrix o, matrix e, matrix t, matrix a)
{
 return  GerumoFunkcija(o, e, t, a);
}
//---------------------------------------------------------------------------
int Optimizavimas::TvarkarascioIlgis()
{
 return  GerumoFunkcija(DO, DE, T, DApr);
}
//---------------------------------------------------------------------------
matrix Optimizavimas::GrazintiDE()
{
 return DE;
}
//---------------------------------------------------------------------------
```

Vaizdavimas.h

```
//---------------------------------------------------------------------------

#ifndef VaizdavimasH
#define VaizdavimasH

#include <ExtCtrls.hpp>
#include "matrix.h"
#include "Gerumas.h"

//---------------------------------------------------------------------------
class Vaizdavimas
{
   private:
    TPaintBox * PaintBoxas;
    TPaintBox * PaintBoxas2;

    int J, M;
    matrix DO; // darbas-operacijos laikas
    matrix DM; // darbas-masinos nr
    matrix DE; // darbas-eiliskumas (pagal masinos nr is eiles)
    matrix DApr; // dienu apribojimai
    int AprS; // dienu apribojimu skaicius
    matrix T; //sprendinys
    matrix L; //vaizdas (matricinis)
    int trukme;

    int * remelisJ;
    int * fonasJ;
    int * remelisM;
    int * fonasM;

   public:
    Vaizdavimas(TPaintBox *,TPaintBox *);
    void DuomenysVaizdavimui(int, int, int, matrix, matrix, matrix, matrix, matrix);
    void DuomenysVaizdavimui(matrix);
    void PiestiM();
    void PiestiJ();
    void GeneruotiNaujasSpalvasM();
    void GeneruotiNaujasSpalvasJ();

};

#endif
```

Vaizdavimas.cpp

```
//---------------------------------------------------------------------------


#pragma hdrstop

#include "Vaizdavimas.h"

//---------------------------------------------------------------------------

#pragma package(smart_init)

//---------------------------------------------------------------------------
Vaizdavimas::Vaizdavimas(TPaintBox * PaintB, TPaintBox * PaintB2)
{
```

```
 PaintBoxas = PaintB;
 PaintBoxas2 = PaintB2;
}
//---------------------------------------------------------------------------
void Vaizdavimas::DuomenysVaizdavimui(int JJ,int MM, int AP,matrix DOO,matrix DMM,matrix DEE,matrix TT,
matrix DAA)
{
 J = JJ;
 M = MM;
 DO = DOO;
 DM = DMM;
 DE = DEE;
 DApr = DAA;
 AprS = AP;
 T = TT;
 Gerumas * Ger = new Gerumas();
 L =  Ger->Vaizdas(DO, DE, T, DApr);
 int Laik = 0;
 for (int i = 1; i <= J; i++)
  if(Laik < L(i,M)) Laik = L(i,M);
 trukme = Laik;
 remelisJ = new int[M];
 fonasJ = new int[M];
 remelisM = new int[J];
 fonasM = new int[J];
 for (int i = 0; i < M; i++)
 {
  remelisJ[i] = random(14777215) + 1000000;
  fonasJ[i] = random(14777215) + 1000000;
 }
 for (int i = 0; i < J; i++)
 {
  remelisM[i] = random(14777215) + 1000000;
  fonasM[i] = random(14777215) + 1000000;
 }
}
//---------------------------------------------------------------------------
void Vaizdavimas::DuomenysVaizdavimui(matrix TT)
{
 T = TT;
 Gerumas * Ger = new Gerumas();
 L =  Ger->Vaizdas(DO, DE, T, DApr);
 int Laik = 0;
 for (int i = 1; i <= J; i++)
  if(Laik < L(i,M)) Laik = L(i,M);
 trukme = Laik;
}
//---------------------------------------------------------------------------
void Vaizdavimas::PiestiM()
{
 int eiluciu = J/10;
 if (J != eiluciu*10) eiluciu++;
 int XXX = PaintBoxas2->Width;
 int YYY = PaintBoxas2->Height - 25*eiluciu;
 int FontDyd = 10;
 PaintBoxas2->Canvas->Font->Color = clBlack;
 PaintBoxas2->Canvas->Font->Size = FontDyd;
 int Mplotis = YYY / M;
 PaintBoxas2->Canvas->Pen->Color = clLtGray;
 int jj;
 for( jj = 0; jj < M; jj++)
 {
  PaintBoxas2->Canvas->TextOutA(2, int(jj*Mplotis + Mplotis/2 - FontDyd/2 - 4), "M" + IntToStr(jj+1));
  PaintBoxas2->Canvas->MoveTo( 0 , jj*Mplotis + Mplotis );
  PaintBoxas2->Canvas->LineTo( XXX , jj*Mplotis + Mplotis );
 }
 int YYYY = jj*Mplotis;
 PaintBoxas2->Canvas->Brush->Color = clWhite;
 PaintBoxas2->Canvas->Pen->Color = clBlack;
 PaintBoxas2->Canvas->MoveTo( 29 , 0 );
 PaintBoxas2->Canvas->LineTo( 29 , YYYY );

 float padala = float(trukme) / float(XXX - 30);
 int YY1 = 0;
 int YY2 = 0;

 for (int m = 1; m <= M; m++)
 {
  YY1 = m*Mplotis - int(3*(Mplotis/4));
  YY2 = YY1 + int(2*(Mplotis/4));
```

```
  for (int k = 1; k <= M; k++)
  {
   for (int i = 1; i <= J; i++)
   {
    PaintBoxas2->Canvas->Brush->Color = fonasM[i-1];
    PaintBoxas2->Canvas->Pen->Color = remelisM[i-1];
    if(m == DE(i,k))
    PaintBoxas2->Canvas->Rectangle(int((L(i,k)-DO(i,DE(i,k)))/padala)+30, YY1, int(L(i,k)/padala)+30,
YY2);
   }
  }
 }

 PaintBoxas2->Canvas->Font->Color = clBlack;
 PaintBoxas2->Canvas->Font->Size = FontDyd;
 PaintBoxas2->Canvas->Brush->Color = clWhite;
 PaintBoxas2->Canvas->TextOutA(XXX-40, 0, IntToStr(trukme));
 //int eiluciu = J/10;
 //if (J != eiluciu) eiluciu++;
 for (int i = 1; i <= eiluciu; i++)
 {
  for (int j = (i-1)*10+1; j <= (i-1)*10+10; j++)
  {
   if(j <= J)
   {
    PaintBoxas2->Canvas->Brush->Color = fonasM[j-1];
    PaintBoxas2->Canvas->Pen->Color = remelisM[j-1];
    PaintBoxas2->Canvas->Rectangle( 2+((j-(i-1)*10)-1)*70,(i-1)*25+YYY+2, 23+((j-(i-1)*10)-1)*70, (i-
1)*25+YYY+23);
    PaintBoxas2->Canvas->Brush->Color = clWhite;
    PaintBoxas2->Canvas->TextOutA(29+((j-(i-1)*10)-1)*70, (i-1)*25+YYY+4, "- J" + IntToStr(j));
   }
  }
 }

 PaintBoxas2->Canvas->Pen->Color = clRed;
 for (int k = 2; k <= AprS; k++)
 {
  if(int(DApr(1,k)/padala)+30 <= XXX)
  {
   PaintBoxas2->Canvas->MoveTo( int(DApr(1,k)/padala)+30 , 0 );
   PaintBoxas2->Canvas->LineTo( int(DApr(1,k)/padala)+30 , YYYY );
  }
 }
 PaintBoxas2->Canvas->Pen->Color = clGreen;
 for (int k = 2; k <= AprS; k++)
 {
  if(int(DApr(2,k)/padala)+30 <= XXX)
  {
   PaintBoxas2->Canvas->MoveTo( int(DApr(2,k)/padala)+30 , 0 );
   PaintBoxas2->Canvas->LineTo( int(DApr(2,k)/padala)+30 , YYYY );
  }
 }

}
//---------------------------------------------------------------------------
void Vaizdavimas::PiestiJ()
{
 int eiluciu = M/10;
 if (M != eiluciu*10) eiluciu++;
 int XXX = PaintBoxas->Width;
 int YYY = PaintBoxas->Height - 25*eiluciu;
 int FontDyd = 10;
 PaintBoxas->Canvas->Font->Color = clBlack;
 PaintBoxas->Canvas->Font->Size = FontDyd;
 int Jplotis = YYY / J;
 PaintBoxas->Canvas->Pen->Color = clLtGray;
 int jj;
 for( jj = 0; jj < J; jj++)
 {
  PaintBoxas->Canvas->TextOutA(2, int(jj*Jplotis + Jplotis/2 - FontDyd/2 - 4), "J" + IntToStr(jj+1));
  PaintBoxas->Canvas->MoveTo( 0 , jj*Jplotis + Jplotis );
  PaintBoxas->Canvas->LineTo( XXX , jj*Jplotis + Jplotis );
 }
 int YYYY = jj*Jplotis;
 PaintBoxas->Canvas->Brush->Color = clWhite;
 PaintBoxas->Canvas->Pen->Color = clBlack;
 PaintBoxas->Canvas->MoveTo( 29 , 0 );
 PaintBoxas->Canvas->LineTo( 29 , YYYY );
```

```
 float padala = float(trukme) / float(XXX - 30);
 int YY1 = 0;
 int YY2 = 0;


 for (int i = 1; i <= J; i++)
 {
  YY1 = i*Jplotis - int(3*(Jplotis/4));
  YY2 = YY1 + int(2*(Jplotis/4));
  for (int k = 1; k <= M; k++)
  {
   PaintBoxas->Canvas->Brush->Color = fonasJ[DE(i,k)-1];
   PaintBoxas->Canvas->Pen->Color = remelisJ[DE(i,k)-1];
   PaintBoxas->Canvas->Rectangle(int((L(i,k)-DO(i,DE(i,k)))/padala)+30, YY1, int(L(i,k)/padala)+30,
YY2);
  }
 }

 PaintBoxas->Canvas->Font->Color = clBlack;
 PaintBoxas->Canvas->Font->Size = FontDyd;
 PaintBoxas->Canvas->Brush->Color = clWhite;
 PaintBoxas->Canvas->TextOutA(XXX-40, 0, IntToStr(trukme));
 //int eiluciu = J/10;
 //if (J != eiluciu) eiluciu++;
 for (int i = 1; i <= eiluciu; i++)
 {
  for (int j = (i-1)*10+1; j <= (i-1)*10+10; j++)
  {
   if(j <= M)
   {
    PaintBoxas->Canvas->Brush->Color = fonasJ[j-1];
    PaintBoxas->Canvas->Pen->Color = remelisJ[j-1];
    PaintBoxas->Canvas->Rectangle( 2+((j-(i-1)*10)-1)*70,(i-1)*25+YYY+2, 23+((j-(i-1)*10)-1)*70, (i-
1)*25+YYY+23);
    PaintBoxas->Canvas->Brush->Color = clWhite;
    PaintBoxas->Canvas->TextOutA(29+((j-(i-1)*10)-1)*70, (i-1)*25+YYY+4, "- M" + IntToStr(j));
   }
  }
 }
 PaintBoxas->Canvas->Pen->Color = clRed;
 for (int k = 2; k <= AprS; k++)
 {
  if(int(DApr(1,k)/padala)+30 <= XXX)
  {
   PaintBoxas->Canvas->MoveTo( int(DApr(1,k)/padala)+30 , 0 );
   PaintBoxas->Canvas->LineTo( int(DApr(1,k)/padala)+30 , YYYY );
  }
 }
 PaintBoxas->Canvas->Pen->Color = clGreen;
 for (int k = 2; k <= AprS; k++)
 {
  if(int(DApr(2,k)/padala)+30 <= XXX)
  {
   PaintBoxas->Canvas->MoveTo( int(DApr(2,k)/padala)+30 , 0 );
   PaintBoxas->Canvas->LineTo( int(DApr(2,k)/padala)+30 , YYYY );
  }
 }
}
//---------------------------------------------------------------------------
void Vaizdavimas::GeneruotiNaujasSpalvasM()
{
 for (int i = 0; i < J; i++)
 {
  remelisM[i] = random(14777215) + 1000000;
  fonasM[i] = random(14777215) + 1000000;
 }
}
//---------------------------------------------------------------------------
void Vaizdavimas::GeneruotiNaujasSpalvasJ()
{
 for (int i = 0; i < M; i++)
 {
  remelisJ[i] = random(14777215) + 1000000;
  fonasJ[i] = random(14777215) + 1000000;
 }
}
//---------------------------------------------------------------------------
```