**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS**

**TAIKOMOSIOS MATEMATIKOS KATEDRA**

**Edgaras Šakurovas**

# GENETINIO IR TABU PAIEŠKOS ALGORITMŲ NAUDOJIMO GAMYBINIŲ TVARKARAŠČIŲ SUDARYMUI ANALIZĖ

Magistro darbas

**Vadovas**
**doc. dr. N. Listopadskis**

**KAUNAS, 2008**

**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS**

**TAIKOMOSIOS MATEMATIKOS KATEDRA**

**TVIRTINU**
**Katedros vedėjas**
**doc. dr. N.Listopadskis**

**2008 06 09**

# GENETINIO IR TABU PAIEŠKOS ALGORITMŲ NAUDOJIMO GAMYBINIŲ TVARKARAŠČIŲ SUDARYMUI ANALIZĖ

Taikomosios matematikos magistro baigiamasis darbas

**Vadovas**
**( ) doc. dr. N. Listopadskis**
**2008 06 05**

**Recenzentas**
**( ) doc. dr. A. Misevičius**
**2008 06 05**

**Atliko**
**FMMM-6 gr. stud.**
**( ) E. Šakurovas**
**2008 06 05**

**KAUNAS, 2008**

**KVALIFIKACINĖ KOMISIJA**

| | |
|---|---|
| **Pirmininkas:** | Leonas Saulis, profesorius (VGTU) |
| **Sekretorius:** | Eimutis Valakevičius, docentas (KTU) |
| **Nariai**: | Algimantas Jonas Aksomaitis, profesorius (KTU) |
| | Vytautas Janilionis, docentas (KTU) |
| | Vidmantas Povilas Pekarskas, profesorius (KTU) |
| | Rimantas Rudzkis, profesorius (MII) |
| | Zenonas Navickas, profesorius (KTU) |
| | Arūnas Barauskas, UAB „Elsis“ generalinio direktoriaus pavaduotojas |

**Šakurovas E. Analysis of Usage of Genetic and Tabu Search Algorithms in Shop Scheduling: Master's work in applied mathematics / supervisor dr. assoc. Prof N. Listopadskis; Department of Applied mathematics, Faculty of Fundamental Sciences, Kaunas University of Technology. – Kaunas, 2008. – 61 p.**

# SUMMARY

A wide area of scheduling problem is industrial so called shop scheduling, which has important applications in real world industrial problems. Job Shop, Open Shop and Flow Shop problems are the most interesting and changeling classes of industrial scheduling at this moment. General problem specification could be specified as follows: there is set of jobs and set of machines, which should interact with each other in some specific way. Typically these problems are hard to solve in traditional methods, such as exhaustive search and etc. Metaheuristics algorithms (Genetic and Tabu search algorithms in this case) seem to be one of the best candidates for finding nearby-optima in proper time. Genetic algorithms are a part of evolutionary computing, which is a rapidly growing area of artificial intelligence; while Tabu search is a leader in local search techniques.

Both algorithms are not easy to implement. Genetic algorithms have lot of sensitive parameters, while Tabu search is wide (sometimes too wide) area of decision of implementation and strongly belongs on problem understanding.

In this work we implemented several genetic algorithms (separated by values of their parameters) and several Tabu search algorithms (separated by neighborhood of solution). Some strategies of genetic algorithms are suggested as conclusion of genetic algorithm parameter research (see [18], [35] for more details). Finally, eight algorithms (genetic with stable, intensive and dynamic strategies, simple tabu, tabu with enlarged diversification, and tabu with differential selection, partial (exact) search and random) are examined for random shop scheduling problems in terms of initial solutions, initial minimum, gained solutions, gained minimum, processing time and difference between initial and gained solutions.

In the end, author concludes, that one kind of genetic parameters (crossover and mutation rates) are especially demanding in sense of algorithm convergence, diversification and intensification aspects, other (number of iterations and population size) should depend on resources, third (elitism) is good "buffers". And finally, while with its simplest form, Tabu search seems to be less competitive in algorithm effectiveness research, its dynamic modification outperforms all proposed genetic algorithms, but both – tabu search with enlarged diversification and dynamic parameter strategy of genetic algorithm – performs quite well and requires more solid future studies to compete with other advanced metaheuristics.

# TURINYS

# LENTELIŲ SĄRAŠAS

# PAVEIKSLŲ SĄRAŠAS

# 1. ĮVADAS
## 1.1. NAGRINĖJAMA PROBLEMA

Tvarkaraščių sudarymo problema šiame industriniame amžiuje yra išties aktuali: traukinių maršrutų tvarkaraštis, darbo valandų tvarkaraštis, materialių resursų (tokių kaip uždirbtų pinigų išleidimo tvarkaraštis) tvarkaraščiai bei pan.

Šiame darbe nagrinėsime gamybinius tvarkaraščius: darbų fabriko (kiekvienai užduočiai atskira darbų tvarka), srautinio fabriko (visiems darbų atlikimo mašinose tvarka vienoda) ir atvirojo fabriko (nėra atlikimo tvarkos apribojimų) problemas. Ir nors bus nagrinėjami pakankamai abstraktūs tvarkaraščiai, kurie realiame pasaulyje retai kur pritaikomi, tačiau parodysime, jog netgi jų klasė yra sudėtinga ne tik didele sprendinio paieškos sritimi, bet ir pačiu algoritmo ir sprendinio įgyvendinimu bei suvokimu.

Industriniame tvarkaraščių sudaryme problema siejasi su darbais ir jų atlikimo galimybėmis. Pavyzdžiui, šiame darbe nagrinėjama darbų fabriko problema formaliai gali būti apibūdinta kaip $n$ įvairių darbų atlikimas su įvairiomis mašinomis $m$, laikantis tam tikro iš anksto nustatyto darbų tose mašinose atlikimo eiliškumo (plačiau žr. [5], [31], [38]).

Ši problema pateikia iškart du konfliktinius aspektus, kuo laisvesni apribojimai, tuo didesnė variacijos galimybė, t.y. iškart plečiasi sprendinio (šiuo atveju optimalaus tvarkaraščio) paieškos sritis, o kuo griežtesni apribojimai, tuo sunkiau įgyvendinti tvarkaraščio optimalumo nustatymą.

Todėl dauguma tvarkaraščio sudarymo problemų negali būti išspręstos tiesioginiais arba taip vadinamais jėgos metodais (pavyzdžiui, visų sprendinių išnagrinėjimu). Visų pirmą dėl per didelių laiko ir skaičiavimo technikos resursų sąnaudų. Čia atrodo labiau tinkamesni tie metodai, kurie gali užkoduoti tvarkaraštį-sprendinį į labiau prieinamą arba lengviau išsprendžiamą formą, tuomet toje erdvėje atlikti atitinkamus tyrimus ir sugrąžinti (atkoduoti) optimalų sprendinį, neprarandant abipusio vienareikšmiškumo.

## 1.2. SPRENDIMO METODAI

Praeito skyrelio pabaigoje minėtų metodų savybėmis pasižymi metaeuristiniai metodai (plačiau žr. [3], [20], [21], [23], [25], [32]): atkaitinimo modeliavimas, tabu paieška, skruzdžių kolonijos optimizavimas, evoliucinės strategijos ir kt. Pastarųjų vienos klasės – genetinių algoritmų – metodais ir bandysime išspręsti gamybinio tvarkaraščio sudarymo problemą. Ir, nors atlikta tikrai nemažai genetinių algoritmų studijų, iki šiol pilnai nėra įrodyta, kodėl genetiniai algoritmai, paremti kryžminimo, mutacijos, paveldėjimo ir kitai operatoriais, veikia tokio pobūdžio problemų sprendime (plačiau žr. [12], [15], [17], [26], [33]). Tai pagrindinis kriterijus, kodėl pasirinkti genetiniai algoritmai, tačiau pasirinkti ne vien jie. Jei genetinis algoritmas intriguoja pačiu savo principu, tai tabu

paieškos algoritmas intriguoja geromis rekomendacijomis (plačiau žr. [6], [7], [8], [9], [10], [11], [14], [16], [22], [24], [29]).

## 1.3. TYRIMAS IR SPRENDIMAS

Šiame darbe buvo tirta parametrų parinkimo įtaka kanoniniam genetiniam algoritmui (ruletės rato atranka, vieno taško individų kryžminimas, dviejų elementų sukeitimo mutacija). Varijuojant įvairias parametrų reikšmes, stebėtas jų poveikis algoritmo veikimui sprendžiant skirtingas tvarkaraščių sudarymo problemas. Autorius nesiekė sukurti „super-turbo-meta“ algoritmą, veikiantį geriau nei visi likusieji iki šiol. Visoms problemoms tai padaryti yra neįmanoma, tuo galima įsitikinti „No free lunch“ teoremų pagalba (plačiau žr. [30]). Tačiau atlikus tyrimus, pastebėtos tam tikros tendencijos, kurios leido modifikuoti tiek genetinį, tiek tabu paieškos algoritmą akivaizdžiai pagerinant jų veikimą.

Gautos genetinio algoritmų parametrų reikšmės suskirstytos į tam tikras klases, kuriomis remtasi kuriant algoritmų strategijas. Pastarosios buvo lyginamos su paprastuoju ir padidintos diversifikacijos tabu paieškos algoritmais, atsitiktinio klaidžiojimo ir dalinės (sprendinio erdvės) tiksliosios paieškos algoritmais.

# 2. TEORINĖ DALIS

## 2.1. KOMBINATORINIS OPTIMIZAVIMAS IR METAEURISTINIAI ALGORITMAI

Kombinatorinis optimizavimas – palyginus su kitomis jauna matematikos atšaka. Išsivysčiusi XX a. antroje pusėje kartu su skaičiuojamosios technikos raida ir sparčiai besivystanti iki šių dienų. Lietuvoje dar mažai paplitusi, tačiau jau žengianti pirmuosius žingsnius (plačiau žr. [20], [21], [22], [23], [24], [25], [27], [28], [34]).

Išsamus ir įvairiapusis įvadas į kombinatorinį optimizavimą, jo problemų sprendimo algoritmus, jų savybes, galimas naujas koncepcijas pateikiamas [3] publikacijoje. Formuluodami pagrindinius teiginius šiame skyrelyje remsimės būtent ja. Šioje publikacijoje teigiama, kad daug optimizavimo problemų svarbių tiek praktiniu, tiek teoriniu požiūriu susideda iš „geriausios“ kintamųjų aibės konfigūracijos paieškos, atitinkančios tam tikrą tikslą. Atrodytų, jog jos natūraliai pasidalina į dvi kategorijas: problemos, kur sprendiniai užkoduojami realiųjų skaičių kintamaisiais ir problemos, kurių sprendiniai užkoduojami diskrečiais kintamaisiais. Tarp pastarosios yra problemų klasė, kuri vadinama kombinatoriniu optimizavimu.

Minėtoje publikacijoje kombinatorinio optimizavimo problema $P = (S, f)$ formaliai apibrėžiama taip:

- kintamųjų aibė $X = \{x_1, ..., x_n\}$;
- kintamųjų sritys $D_1, ..., D_n$;
- kintamųjų apribojimai;
- minimizuojama tikslo funkcija $f$, kur $f : D_1 \times ... \times D_n \to R^+$.

Visų galimų įmanomų priskyrimų aibė yra $S = \{s = \{(x_i, v_i), ..., (x_n, v_n)\} \mid v_i \in D_i,\ s \text{ tenkina visus apribojimus}\}$. $S$ paprastai vadinama paieškos (arba sprendinio) erdve, kadangi kiekvienas aibės elementas gali būti laikomas sprendiniu-kandidatu. Tam, kad išspręstumėme kombinatorinio optimizavimo problemą, turime rasti $s^* \in S$ su minimalia tikslo funkcijos reikšme tokia, kad $f(s^*) \leq f(s)\ \forall s \in S$. $s^*$ vadinamas globaliai optimaliu sprendiniu iš $(S, f)$ ir aibė $S^* \subseteq S$ vadinama globaliai optimalių sprendinių aibė.

Kombinatorinio optimizavimo problemų pavyzdžiai yra keliaujančio komivajožieriaus problema, kvadratinio paskirstymo problema, tvarkaraščių ir grafikų problemos ir t.t. Dėl praktinės kombinatorinio optimizavimo svarbos buvo išvystyta daug algoritmų jų sprendimui. Šie algoritmai klasifikuojami į pilnutinius arba aproksimuojančius algoritmus. Pilnutiniai algoritmai garantuoja kiekvieno kombinatorinio optimizavimo problemos baigtinio dydžio atvejo optimalaus sprendinio radimą per ribotą laiką. Vis dėlto kombinatorinio optimizavimo problemoms, kurios yra $NP$-sunkios,

neegzistuoja polinominio laiko algoritmas, tariant, kad $P \neq NP$. Todėl pilnutiniams metodams blogiausiu atveju gali reikėti eksponentinio skaičiavimo laiko. Tai dažnai veda prie per ilgo skaičiavimo laiko praktiniams tikslams. Tokiu būdu, dėmesys aproksimuojantiems metodams kombinatorinio optimizavimo problemoms spręsti per paskutinius 30 metų žymiai išaugo. Aproksimuojančiuose algoritmuose aukojama optimalaus sprendinio radimo garantija žymiai sumažinto laiko kiekio labui.

Tarp pagrindinių aproksimacijos metodų paprastai išskiriamos dvi klasės: konstruktyvių metodų ir lokalios paieškos metodų. Konstruktyvūs algoritmai generuoja sprendinius nuo pradžių vis pridėdami iš pradžių tuščiam daliniam sprendiniui komponentus, ligi sprendinys užpildomas. Jie paprastai yra greičiausi aproksimuojantys metodai, visgi jie grąžina prastesnės kokybės sprendinius palyginus su vidinės paieškos algoritmais. Lokalios paieškos algoritmai pradeda nuo tam tikro pradinio sprendinio ir iteracijomis pirmyn bando pakeisti darbinį sprendinį geresniu sprendiniu pagal atitinkamai apibrėžtą darbinio sprendinio aplinką, kur aplinka formaliai aprašoma tokiu būdu:

Aplinkos struktūra yra funkcija $N : S \to 2^S$, kuri kiekvienam $s \in S$ priskiria aplinkos aibę $N(s) \subseteq S$. $N(s)$ vadinama $s$ aplinka.

Aplinkos struktūros įvedimas leidžia apibrėžti lokaliai minimalių sprendinių sąvoką.

Lokaliai minimalus sprendinys (arba lokalus minimumas) atsižvelgiant į struktūrą $N$ yra sprendinys $\hat{s}$ toks, kad $\forall s \in N(\hat{s}) : f(\hat{s}) \leq f(s)$. $\hat{s}$ vadiname griežtai lokaliai minimaliu sprendiniu, jei $f(\hat{s}) \leq f(s)\ \ \forall s \in N(\hat{s})$.

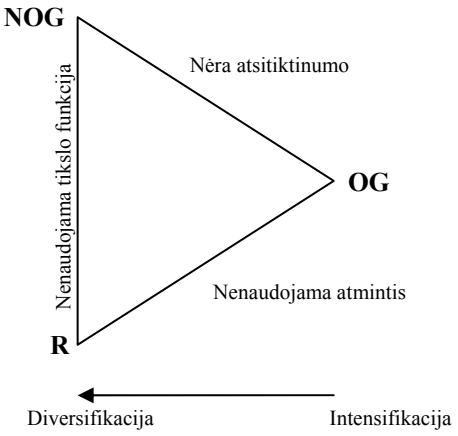
Per paskutinius 20 metų iškilo nauja aproksimacijos algoritmų rūšis, iš esmės bandanti sujungti pagrindinius euristinius metodus aukštesniame lygmeny. Dabar šie metodai paprastai vadinami metaeuristika (euristika kilusi iš graikiško žodžio *heuriskein*, kuris reiškia „ieškoti“ ir priedėlis meta, kuris reiškia „virš, aukštesniame lygyje“). Šiai algoritmų klasei priklauso skruzdžių kolonijos optimizavimas, evoliuciniai skaičiavimai, įskaitant ir genetinius algoritmus, iteracinė lokali paieška, atkaitinimo modeliavimas ir tabu paieška.

Metaeuristikų savybes galima apibūdinti tokiu būdu:

- metaeuristikos yra strategijos, kurios „vadovauja“ paieškos procesui;
- tikslas yra efektyviai tirti paieškos erdvę tam, kad būtų galima rasti (artimą) optimalų sprendinį;
- metodai, kurie yra metaeuristiniai skiriasi tarpusavyje nuo paprastos lokalios paieškos procedūros iki sudėtingų mokymosi procesų;
- metaeuristiniai algoritmai yra apytiksliai ir paprastai nedeterminuoti;
- jos gali įtraukti į sudėtį mechanizmus tam, kad išvengtų užstrigimų uždarytoje paieškos erdvės srityje;

- pagrindinės metaeuristikų koncepcijos leidžia abstraktaus lygio aprašymus;
- metaeuristikos nėra specifinės problemai;
- metaeuristikos gali pasinaudoti specifinės srities žiniomis euristikų formoje, kurios yra valdomos aukštesnio lygio strategija;
- šiandien daug patobulintų metaeuristikų naudoja paieškos patirtį (įgyvendintą tam tikroje atminties formoje) tam, kad vadovautų paieškai.

Įdomus ir gana novatoriškas požiūris į metaeuristikų klasifikavimą taip pat pateikiamas [3] publikacijoje. Čia išryškinama dviejų sąvokų – *intensifikacijos* ir *diversifikacijos* svarba metaeuristiniams algoritmams – tai yra taip vadinama *I&D sistema* (2.1 pav.). Galime pasinaudoti vienu iš daugelio publikacijoje pasiūlytų šių sąvokų apibrėžimų, jog intensifikacija – tai atsargiai ir intensyviai ieškoti aplink gerus sprendinius, rastus praeitoje paieškoje. Diversifikacija, atvirkščiai, yra valdyti paiešką nelankytose srityse. Kitais žodžiais tariant, įvairios metaeuristinės idėjos turi būti suprantamos šių dviejų sąvokų požiūriu ir metaeuristiniai algoritmai turi būti sukonstruojami taip, kad intensifikacija ir diversifikacija vaidintų subalansuotą vaidmenį. 2.1 pav. matome I&D sistemos jungtinį vaizdą intensifikacijos ir diversifikacijos atžvilgiu metaeuristikose. Ten *OG* – I&D komponentai valdomi tiktai tikslo funkcija, *NOG* – I&D komponentai valdomi tiktai viena ar daugiau funkcijų, skirtingų nei tikslo funkcija, *R* – I&D komponentai valdomi tiktai atsitiktinumu.




**2.1 pav. I&D sistemos schema [3]**

Vadinasi, tinkamas balansas tarp intensifikacijos ir diversifikacijos yra reikalingas tam, kad gautume efektyvią metaeuristiką. Be to, šis balansas neturi būti fiksuotas arba keistis tik į vieną pusę (pvz., tolygiai didinti tik intensifikaciją). Mūsų nagrinėjamų algoritmų I&D komponentai pateikti 2.1 lentelėje.

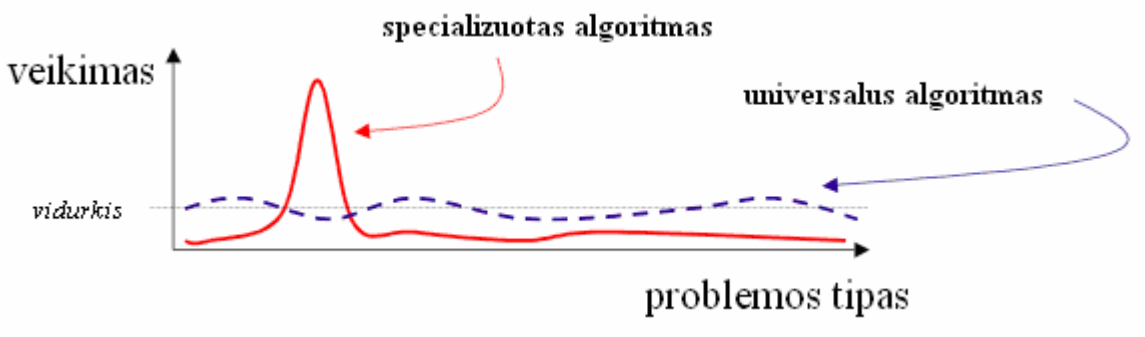
**2.1 lentelė.**

**I&D komponentai genetiniam ir tabu paieškos algoritmams [3]**

| Metaeuristika | I&D komponentai |
|---|---|
| Tabu paieška | aplinkos parinkimas (tabu sąrašai) |
| | aspiracijos kriterijus |
| Evoliuciniai skaičiavimai | rekombinacija |
| | mutacija |
| | atranka |

Nors metaeuristikos efektyviai sprendžia daugelį kombinatorinio optimizavimo problemų, tačiau verta paminėti kelis paradoksus, taip vadinamas „No free lunch“ teoremas (2.2 pav.) paieškoje/optimizavime [30]:

- „... visi algoritmai, kurie ieško įverčio funkcijos ekstremumų veikia tiksliai vienodai, kuomet imamas vidurkis visoms galimoms įverčio funkcijoms. Ypač jei algoritmas A veikia geriau nei B tam tikrose įverčio funkcijose, tuomet, trumpai tariant, turi egzistuoti tiksliai tiek pat kitų funkcijų, kuriuose B veikia geriau ne A.“
- „…bet kuriam (optimizavimo) algoritmui bet kuris jo veikimo pagerinimas vienai problemų klasei yra kaip tik pabloginimas kitai uždavinių klasei.“




**2.2 „No free lunch“ teoremų grafinė iliustracija [30]**

## 2.2. TVARKARAŠČIŲ SUDARYMO PROBLEMA

Tvarkaraščių sudarymo tikslas yra pagaminti tvarkaraštį tam tikrai situacijai. Problemoje išdėstoma, kas turi užimti vietą, o sprendimas tiksliai apibūdina, kas turi atsitikti tuo metu.

Šios problemos iškyla gamyklose, personalo planavime, aerodromuose, geležinkelių stotyse ir t.t.

Šiame darbe nagrinėkime pakankamai abstrakčias (neprarasdami problemos sudėtingumo) tvarkaraščių sudarymo problemas.

Nagrinėkime tam tikrą gamyklą, kurioje resursai vadinami mašinomis. Šios mašinos atlieka operacijas darbams. Kiekvienas darbas susideda iš operacijų skaičiaus. Galima darbą įsivaizduoti kaip mašiną, kuriai reikia pataisymų skaičiaus arba kaip studentą, kuriam reikia dalyvauti tam tikrame skaičiuje klasės susitikimų.

## 2.2.1. APIBRĖŽIMAS

Tvarkaraščių sudarymo uždaviniai siejasi su resursų paskirstymu per laiką tam, kad būtų galima atlikti užduočių seką. Kiekvieną tvarkaraščių sudarymo problemą charakterizuoja trys komponentai:

- Mašinų skaičius $m$, darbų skaičius $n$, atlikimo laikų kiekvienam darbui kiekvienoje mašinoje matrica $A$.
- Apribojimų aibė $C$, kurie turi būti tenkinami kiekvienam tvarkaraščiui.
- Tikslo funkcija $f$, kurią reikia minimizuoti.

Tvarkaraščių sudarymo problemos įdomios tiek praktiniu, tiek teoriniu požiūriu. Kadangi šią problemą pasirinkome genetinių algoritmų taikymui, tai čia nagrinėkime pakankamai paprastas problemas, kurias lengva apibūdinti. Kadangi tvarkaraščiai yra su daugybe apribojimų (kurie dažniausiai ir yra sutinkami realiame pasaulyje) iškart komplikuojasi ne tik problemos sprendimas, bet ir pats jos formulavimas, nagrinėjimas bei įgyvendinimas. Todėl mūsų abstrakčios problemos formuoja labai abstraktų modelį realaus pasaulio situacijai. Apibrėžimo supaprastinimui tarkime, kad kiekvienas darbas susideda lygiai iš vienos operacijos kiekvienoje mašinoje. Darbų atlikimo laikai duoti kaip $m \times n$ matmenų matrica $A$, kur $A_{i,j}$ yra $j$-tojo darbo atlikimas $i$-tojoje mašinoje ($1 \le i \le m, 1 \le j \le n$). Atlikimo laikai gali būti ir realūs skaičiai. Tvarkaraščių sudarymo problemos tikslas yra surasti tokį tvarkaraštį, kurį pilnai apibūdintų $m \times n$ matmenų matrica $S$, kur $S_{i,j}$ yra pradžios laikas $j$-tojo darbo operacijos $i$-tojoje mašinoje. Tinkami tvarkaraščiai privalo nenaudoti tų pačių darbų arba mašinų vienu metu, taigi visi tvarkaraščiai privalo tenkinti sekančius apribojimus:

$$\forall i, j \; S_{i,j} \ge 0,$$

$$\forall i, j, k \; \left( j \ne k \wedge S_{i,j} \le S_{i,k} \right) \Rightarrow S_{i,j} + A_{i,j} \le S_{i,j},$$

$$\forall i, h, j \; \left( i \ne h \wedge S_{i,j} \le S_{h,j} \right) \Rightarrow S_{i,j} + A_{i,j} \le S_{h,j}.$$

Minimizuojama funkcija $f$ ir suminis tvarkaraščio trukmės ilgis, arba laikas, kuomet baigiasi paskutinė operacija, t.y.

$$f(S) = \max_{i,\, j} \left( S_{i,j} + A_{i,j} \right).$$

Tai dažnai vadinama atlikimo trukmės kriterijum ir tai labai svarbus matas. Alternatyvos yra srauto laikas, vėlavimas ir pavėlavimas, tačiau kai kurios iš šių sąvokų reikalauja papildomos informacijos tokios kaip numatomas pristatymo laikas darbams.

Tokiu būdu apibrėžti tvarkaraščiai sudaro statinių tvarkaraščių klasę. Šioje klasėje visi darbai yra prieinami nuo pradžios.

Dauguma metodų įgyvendintų vienam tvarkaraščių sudarymo variantui gali būti pritaikyti kitam. Pagrindinis skirtumas yra tai, kad atlikimo trukmė yra netinkama sąvoka kito tipo tvarkaraščiams, pavyzdžiui – dinaminiams, kadangi viskas apie problemą yra žinoma ir nesitikima, kad kas nors atsitiktų. Praktikoje mašinos gali sugesti ir visas atlikimo procesas stipriai vėluoti, arba į atlikimo proceso eiliškumą įsiterpti kitas, didesnį prioritetą turintis darbas ir tuomet visą (arba beveik visą) tvarkaraštį reikia suformuoti iš naujo. Tvarkaraščiai, kurie yra tinkami netgi netikėtų atsitikimų atveju, vadinami tvirtais arba lanksčiais tvarkaraščiais. Ši tvarkaraščių sudarymo šaka turi geresnę praktinę vertę, bet jie yra specifiniai atskiroms problemoms, todėl mažiau tinkami etalono naudojimui.

## 2.2.2. PROBLEMOS

Kadangi spręsime tris pagrindinius gamybinių tvarkaraščių sudarymo uždavinius, apribojimams $C$ turime tris skirtingus apibrėžimus:

**Srautinis fabrikas.**

Srautiniame fabrike visi darbai privalo patekti į mašinas nustatyta tvarka. Problemos specifikacijai duota papildomas $n$ ilgio vektorius $B$ stulpelis. Jame išvardinta mašinų aplankymo tvarka, ir kiekvienas darbas turi jos laikytis. Taigi, formalizuota tokiu būdu:

$$S \sim C \Rightarrow \forall i,h,j \ \left(i < h \Rightarrow S_{B_i,j} \le S_{B_h,j}\right).$$

**Darbų fabrikas.**

Darbų fabrike kiekvienas darbas privalo patekti į mašinas nustatyta tvarka. Problemos specifikacijos duota papildoma $m \times n$ matmenų matrica $B$. Kiekvienas matricos $B$ stulpelis yra mašinų permutacija, ir $j$-tasis darbas privalo patekti į mašinas eiliškumo tvarka duota matricos $j$-tuoju stulpeliu. Formaliai:

$$S \sim C \Rightarrow \forall i,h,j \ \left(i < h \Rightarrow S_{B_{i,j},j} \le S_{B_{h,j},j}\right).$$

Taigi, nesunku pastebėti, kad srautinis fabrikas yra atskiras darbų fabriko atvejis.

**Atvirasis fabrikas.**

Atvirajame fabrike jokių papildomų apribojimų nėra, t.y.

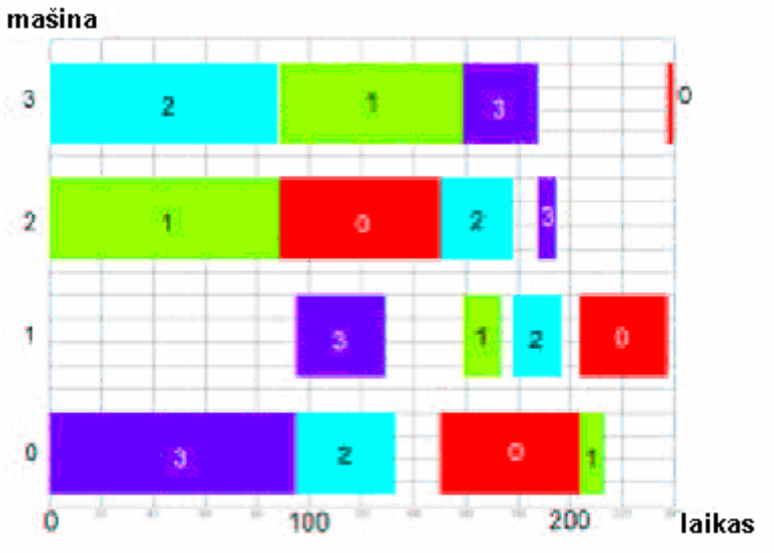
$$C = O.$$

Kiekvienas darbų arba srautinio fabriko tvarkaraštis yra tinkamas atvirojo fabriko tvarkaraščiui. Pastebėsime, kad darbai ir mašinos gali būti lengvai tarpusavyje keičiamos, laikantis ekvivalenčios problemos. Taigi galime įsivaizduoti atvirąjį fabriką kaip šokių varžybų tvarkaraščio sudarymą, kai duota $n$ berniukų ir $m$ mergaičių ir laikai, kuriuos jie nori praleisti drauge.

Dabar pateiksime darbų fabriko pavyzdį [31]. Darbų fabriko problema apibrėžiama kaip dvi matricos $A$ (atlikimo laikams apibrėžti) ir $B$ (mašinų tvarkai kiekvienam darbui apibrėžti). Tai problema su 4 mašinomis ir 4 darbais.

$$A = \begin{pmatrix} 54 & 9 & 38 & 95 \\ 34 & 15 & 19 & 34 \\ 61 & 89 & 28 & 7 \\ 2 & 70 & 87 & 29 \end{pmatrix} \text{ ir } B = \begin{pmatrix} 2 & 4 & 2 & 1 \\ 3 & 3 & 4 & 2 \\ 1 & 1 & 3 & 4 \\ 4 & 2 & 1 & 3 \end{pmatrix}.$$

Šios problemos sprendinys pateiktas 2.3 pav. Toks grafinis tvarkaraščių vaizdavimas vadinamas Ganto diagrama. Kiekviena eilutė atitinka mašiną. Operacijos kiekvienoje mašinoje pavaizduotos kaip spalvoti blokai. $x$ ašyje yra duotas laikas, taigi kiekvienas blokas yra tokio dydžio, kiek atitinkamas darbas, kurio numeris pavaizduotas bloko viduje, užtrunka, nors galimas ir kitoks blokų priskyrimas darbams. Šio tvarkaraščio atlikimo laikas yra 240 vienetų. Ganto diagrama gali būti naudinga grafiniu būdu patikrinti tvarkaraščių sudarymo priemones.




**2.3 pav. Ganto diagramos pavyzdys [31]**

## 2.3. GENETINIS ALGORITMAS

Genetinius algoritmus 1975 metais įvedė Holland [17], nemažai prie jų vystymo prisidėjo Goldberg [12]. Genetinis algoritmas yra paieškos metodas naudojamas kompiuterių moksle apytikslių sprendinių radimui optimizavimo ir paieškos problemose. Genetiniai algoritmai yra atskira klasė evoliucinių algoritmų, kurie naudoja metodus, paremtus evoliucinės biologijos principais, tokiais kaip paveldėjimas, individų mutacija, atranka ir kryžminimas.

Genetiniai algoritmai paprastai įgyvendinami kaip kompiuterinė imitacija, kurioje sprendinių-kandidatų $y$ (vadinamų individais) abstrakčių $c$ ilgio reprezentacijų $x$ (vadinamų chromosomomis)

$p$ dydžio populiacija optimizavimo problemai vystosi link geresnių sprendinių. Evoliucija prasideda nuo visiškai atsitiktinių individų populiacijos ir vyksta kartose. Kiekvienoje kartoje atliekamas visų populiacijos narių įvertinimas, daugialypiai individai stochastiškai išrenkami iš einamosios populiacijos (pagal jų įverčius), modifikuojami (mutuoja ar kryžminami) tam, kad suformuotų naują populiaciją, kuri taps einamąją sekančioje algoritmo iteracijoje.

## 2.3.1. ALGORITMO SCHEMA

Pagrindinio genetinio algoritmo schemą galima pavaizduoti sekančiais žingsniais:

1. **[Pradžia]** Sugeneruoti atsitiktinę populiaciją $P$ iš $p$ chromosomų.
2. **[Įvertinimas1]** Įvertinti populiacijos $P$ kiekvienos chromosomos $x_i$ tinkamumą $e_i = f(x_i)$ $(1 \le i \le p)$.
3. **[Nauja populiacija]** Sukurti naują tokio paties dydžio populiaciją $P'$, kartojant tokius žingsnius, kol neužpildoma visa populiacija:
   - **3.1. [Atranka]** Išrinkti dvi chromosomas-tėvus $x_i$ ir $x_j$ iš populiacijos $P$ atitinkamai pagal jų įverčius $e_i$ ir $e_j$ $(1 \le i \le p, 1 \le j \le p)$.
   - **3.2. [Kryžminimas]** Su kryžminimo tikimybe $\rho_K$ sukryžminti išrinktus tėvus naujos chromosomos-palikuonio $x'_k$ sukūrimui ($k$ – populiacijos $P'$ pildymo žingsnio numeris).
   - **3.3. [Mutacija]** Su mutacijos $\rho_M$ tikimybe pakeisti palikuonį $x'_k$ kiekviename lokuse (chromosomos $i$-tojoj pozicijoj $(1 \le i \le c)$).
   - **3.4. [Priėmimas]** Įtraukti gautą palikuonį $x'_k$ į naująją populiaciją $P'$.
4. **[Pakeitimas]** Pakeisti senąją populiaciją $P$ naująja $P'$.
5. **[Įvertinimas2]** Įvertinti populiacijos $P$ kiekvienos chromosomos $x_i$ tinkamumą $e_i = f(x_i)$ $(1 \le i \le p)$.
6. **[Testas]** Jei tenkinamas pabaigos kriterijus, sustabdyti algoritmą ir grąžinti geriausią populiacijos $P$ sprendinį.
7. **[Ciklas]** Grįžti į 2 žingsnį.

Kaip matome, pagrindinio genetinio algoritmo schema yra labai bendra. Todėl daug dalykų gali būti įgyvendinti skirtingai įvairiose problemose. Čia iškyla tokie klausimai:

- Kaip sukurti chromosomas, t.y. kokį kodavimo tipą pasirinkti? Nes su šia problema siejasi du pagrindiniai genetinių algoritmų reprodukciniai operatoriai – kryžminimas ir mutacija.

- Kaip atrinkti chromosomas-tėvus kryžminimui? Tai gali būti atliekama įvairiausiais būdais, visgi pagrindinė idėja yra ta, kad reikia išrinkti geresnius tėvus tikintis, kad geresni tėvai duos geresnius palikuonis.
- Kaip parinkti tinkamus algoritmo parametrus?
- Galiausiai, ar naujoj populiacijoj $P'$, kuri sudaryta tik iš naujų sprendinių neprarasime geriausio sprendinio iš buvusios $P$?

## 2.3.2. REPREZENTACIJA

Tipiniam genetiniam algoritmui būtina apibrėžti du dalykus:

1) genetinę sprendinių $y$ reprezentaciją $x$;

2) įverčio funkciją $f(x)$ chromosomų (sprendinių) įvertinimui.

Pirmoji problema – chromosomų (sprendinio) kodavimas – yra vienas iš pradinių uždavinių taikant genetinius algoritmus problemų sprendimui. Kodavimas priklauso nuo problemos specifikos, todėl galimų kodavimo variantų yra išties daug ir įvairių. Pats paprasčiausias kodavimo būdas yra dvejetainis. Teoriškai, mažesnis alfabetas, geresnis našumas, tačiau paradoksalu, kad geri rezultatai buvo gauti naudojant ir kitus kodavimus, pavyzdžiui realių skaičių chromosomas.

Permutacinis arba kėlinių kodavimas naudingas eiliškumo nustatymo problemoms. Permutaciniame kodavime kiekviena chromosoma $x$ yra numerių eilutė, kuri reprezentuoja numerį tam tikroje sekoje.

| | |
|---|---|
| **Chromosoma1** | 1 2 4 5 9 8 7 6 3 |
| **Chromosoma2** | 6 5 4 7 8 9 1 3 2 |

**2.4 pav. Permutacinio chromosomų kodavimo pavyzdys**

Antroji problema – sprendinio įvertinimo (arba tiesiog įverčio) funkcijos įgyvendinimas ir yra svarbus faktorius tiek algoritmo greičiui, tiek jo efektyvumui. Įverčio funkcija apibrėžiama per genetinę reprezentaciją ir matuoja reprezentuojamo sprendinio kokybę. Vadinasi, įverčio funkcija turi vienareikšmiškai atkoduoti ne tik chromosomą, bet ir tinkamai įvertinti jos kokybę. Todėl bendru atveju įgyvendinti įverčio funkciją yra tiek pat sunku kaip ir atlikti chromosomos reprezentaciją.

Tačiau chromosomos įvertinimas vien tik įverčio funkcija tam tikroms problemoms nelabai tinka, nes gali blogai sąlygoti tiek patį algoritmą, tiek ir sprendinį, todėl dažnai taikoma taip vadinama chromosomų pajėgumo funkcija, kuri $i$-tajai chromosomai apibrėžiama

$$F(x_i) = f(x_i) / \bar{f},$$

čia $\bar{f} = \frac{1}{p}\sum_{i=1}^{p} f(x_i)$ yra populiacijos chromosomų įverčių vidurkis, o $p$ - populiacijos dydis.

## 2.3.3. REPRODUKCIJA

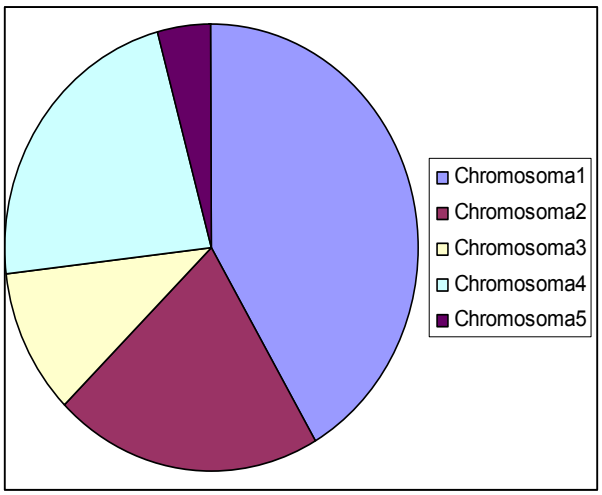
Jei turime apibrėžę genetinę reprezentaciją ir įverčio funkciją, genetinis algoritmas tęsiamas inicijuojant atsitiktinių sprendinių populiaciją $P$, tuomet tobuliname ją kartodami atrankos, kryžminimo ir mutacijos operatorių taikymą.

Iš pradžių daug atskirų sprendinių atsitiktinai sugeneruojami pradinės populiacijos suformavimui. Populiacijos dydis $p$ priklauso nuo problemos pobūdžio, bet paprastai susideda iš keleto šimtų ar tūkstančių galimų sprendinių (nors kartais užtenka chromosomos ilgio populiacijos, t.y. $p = c$). Tradiciškai, populiacija sugeneruojama atsitiktinai, kiek įmanoma padengiant visą galimų sprendinių sritį (paieškos erdvę), kadangi algoritmas tam tikrais atvejais, kurių yra nemažai, gali nuo to priklausyti.

Kiekvienoje sekančioje epochoje, egzistuojančios populiacijos $P$ proporcija atrenkama naujos kartos kūrimui, t.y. naujos populiacijos $P'$ formavimui. Atskiri sprendiniai atrenkami įverčių paremtu proceso metu, kur geriau įvertinti sprendiniai (pagal įverčio arba kurią nors kitą funkciją) yra paprastai labiau galimi išrinkimui. Tai paremta Darvino evoliucijos teorija, kad geriausieji turi išlikti ir daugintis toliau.

Yra daugybė metodų, kaip išrinkti geriausias chromosomas kryžminimui. Čia pateiksime vieną iš įprastų – ruletės rato atranką.

Chromosomos-tėvai išrenkami atitinkamai pagal jų įverčius. Geresnės chromosomos turi daugiau šansų būti atrinktos. Įsivaizduokime ruletės ratą, kuriame išdėstytos visos populiacijos chromosomos, kiekviena turi savo vietą atitinkamai pagal jos įvertį, kaip pavaizduota 2.5 pav. Tada metamas kamuoliukas ir atrenkama chromosoma. Akivaizdu, kad chromosomos su geresniu įverčiu bus atrinktos daugiau kartų.




**2.5 pav. Ruletės rato atrankos pavyzdys**

Tokią atrinkimo schemą galima pavaizduoti tokiais žingsniais:

1. **[Suma]** Apskaičiuoti visų populiacijos chromosomų pajėgumų $f$ sumą: $S = \sum_{i=1}^{p} f_i$ .
2. **[Generacija]** Sugeneruoti atsitiktinį skaičių $r$ iš intervalo $[0; S)$.
3. **[Ciklas]** Iš eilės nagrinėti populiacijos narius pradedant $k = 1$, ir sumuoti jų pajėgumus $S_k = \sum_{i=1}^{k} f_i$, kai $S_k > r$, tai nario, kurį reikia atrinkti, populiacijoje numeris $k$.

Ruletės rato atranka turi problemų, kai chromosomų pajėgumui daug skiriasi. Pavyzdžiui, geriausios chromosomos pajėgumas yra 80% ruletės rato, tai kitos chromosomos turės labai mažus šansus pakliūti į atranką, tokiu būdu yra tikslinga šiek tiek „iškreipti“ esamą situaciją nedideliu visų atrankoje dalyvaujančių individų pajėgumo funkcijos reikšmės pakėlimu ir taip bent šiek tiek padidinti blogiausių individų atrankos šansus.

Po chromosomų atrankos kiti – kryžminimas ir mutacija – yra du pagrindiniai genetinių algoritmų operatoriai, nuo kurių labai priklauso genetinio algoritmo eiga. Šių operatorių tipas ir įgyvendinimas, žinoma, priklauso nuo pasirinkto kodavimo ir taip pat nuo problemos. Yra daugybė variantų, kaip atlikti chromosomų kryžminimą ir mutaciją. Čia pateiksime keletą, skirtų permutaciniam kodavimui, kai iš dviejų tėvų-chromosomų gaunami du palikuonys-chromosomos.

Ar kryžminti pasirinktas chromosomas-tėvus, nusako kryžminimo tikimybė $\rho_K$, jeigu chromosomų nekryžminame, vadinasi, chromosomos-palikuonis yra tiksli chromosomų-tėvų kopija.

Vieno taško kryžminimas: pasirenkamas vienas kryžminimo taškas eilutėje ir tuomet nuo eilutės pradžios iki to kryžminimo taško eilutės dalis nukopijuojama iš vieno tėvo, o likusi dalis iš kito bei atvirkščiai.

Vienas iš bendriausių mutacijos atvejų yra sukeitimas, kuomet chromosomoje dviejuose atsitiktinai parinktose pozicijose reikšmės tarpusavyje sukeičiamos.

**Chromosomų kryžminimas**

**Vieno taško kryžminimas**

**127**54936 + 8276**91354** = **127896354**

1278**54936** + **8276**91354 = **827654931**

**Chromosomų mutacija**

**Elementų sukeitimas**

1**4**5267**9**83 => 1**9**5267**4**83

**2.6 pav. Permutacinio kodavimo chromosomų kryžminimo ir mutacijos pavyzdžiai**

Kiekvienai naujai sprendinio reprezentacijai $x'$ pagaminti, išrenkama chromosomų-tėvų pora iš darbinės populiacijos $P$. Gaminant chromosomą-palikuonį naudojant prieš tai minėtus kryžminimo ir mutacijos metodus, sukuriami nauji sprendiniai, kurie paprastai paveldi daugybę savo „tėvų“ savybių. Nauji tėvai išrenkami kiekvieniems dvejiems vaikams, ir procesas tęsiamas, kol nesukuriama nauja tinkamo dydžio sprendinių populiacija.

Šie procesai galiausiai baigiasi sekančios kartos chromosomų populiacija, kuri yra skirtinga nei pradinė karta. Bendru atveju chromosomų įverčio vidurkis bus padidintas šia populiacijos procedūra, kadangi tiktai geriausi organizmai iš pirmos kartos atrenkami veisimui, kartu su maža proporcija mažiau tinkamų sprendinių.

Tačiau svarbus klausimas yra, kokia vykdoma atranka. Kadangi gali atsitikti taip, jog sukurdami naują populiaciją $P'$, mes prarasime geriausią sprendinį iš $P$, ir sekančiose epochose galim jo jau nebepasiekti. Todėl šiam atvejui naudojamas taip vadinamas elitizmas. Tai reiškia, kad bent vienas geriausias sprendinys yra nukopijuojamas iš populiacijos $P$ be pakeitimų į naująją $P'$. Taigi visada geriausias sprendinys yra išlaikomas. Bendru atveju gali būti nukopijuota $E$ procentų populiacijos, kai išsaugomas daugiau nei vienas geriausias sprendinys.

### 2.3.4. ALGORITMO PARAMETRŲ NUSTATYMO PROBLEMOS

Atranka yra akivaizdžiai svarbus genetinis operatorius, bet skiriasi nuomonė dėl kryžminimo ir mutacijos svarbos. Iš pirmo žvilgsnio galime teigti, kad kryžminimas yra svarbiausias, o tuo tarpu mutacija yra tiktai būtina potencialių sprendinių nepraradimo užtikrinimui. Tačiau iš kitos pusės kryžminimas didžia dalimi tolygioje populiacijoje tiktai tarnauja dauginimui, o naujovės iš pradžių randamos mutacijos pagalba, ir netolygioje populiacijoje kryžminimas beveik visada lygus didelei mutacijai (kuri tikėtina gali būti katastrofinė).

Labai maža mutacijos tikimybė gali nuvesti prie genetinio dreifo arba pirmalaikio genetinio algoritmo konvergavimo į lokalinį optimumą. Mutacijos tikimybė, kuri yra per didelė gali nuvesti prie gerų sprendinių netekimo. Yra teorinis, bet ne praktinis viršutinis ir apatinis rėžiai šiems parametrams, kurie gali padėti vesti atranką.

Taigi, akivaizdu, kad kiekvienai specifinei problemai yra skirtingos specifinės parametrų savybės. Rekomendacijos parametrams dažnai yra tam tikri empiriniai tyrimai, kurie dažniausiai atliekami su dvejetainiu kodavimu.

Teoriškai kryžminimo tikimybė turėtų būti didelė, apie 80%-95% (vis dėlto tam tikri rezultatai rodo, kad kai kurioms problemoms 80% kryžminimo tikimybė yra geriausia). Iš kitos pusės mutacijos tikimybė turėtų būti labai maža, norint neiškreipti daugumos sprendinių. Manoma, kad geriausia tikimybė yra 0.5%-1%.

Galima stebėtis, kad didelis populiacijos dydis dažniausiai nepagerina genetinio algoritmo veikimo (sprendinio radimo greičio atžvilgiu). Geras populiacijos dydis yra 20-30, tačiau kiti teigia, kad 50-100 yra geriausias populiacijos dydis. Kai kurie tyrimai taip pat rodo, kad geriausios populiacijos dydis priklauso nuo kodavimo, t.y. užkoduotos eilutės ilgio. Galimos genetinio algoritmo parametrų reikšmes pateiktos 2.2 lentelėj.

**2.2 lentelė**

**Trijų matematikų siūlomos parametrų reikšmės [32]**

| Valdymo parametrai | De Jong | Schaffer | Grefenstette |
|---|---|---|---|
| Populiacijos dydis | 50-100 | 20-30 | 30 |
| Kryžminimo tikimybė | 0.60 | 0.75-0.95 | 0.95 |
| Mutacijos tikimybė | 0.001 | 0.005-0.01 | 0.01 |

## 2.4. TABU PAIEŠKOS ALGORITMAS

Tabu paieškos metodą nepriklausomai vienas nuo kito pasiūlė Hansen, Jaumard [14] ir Glover [7] (toliau vystė Glover [9], [10], ir galiausiai savotiška tabu paieškos Biblija yra Glover, Laguna knyga [11]). Šis metodas paremtas lokalios paieškos principais, tačiau turi papildomų komponentų, priešingai nuo lokalios paieškos, neleidžiančių sustoti lokaliame minimume.

Tam, kad paieškos metodas būtų pavadintas tabu paieška, reikalingos trys charakteristikos [29]:

- Tabu: laikini apribojimai, išskiriant tam tikrus pasirinkimus iš paieškos;
- Trumpalaikė atmintis: tabu sąrašas tam, kad apsaugotų nuo grįžimo į ankstesnius žingsnius laiko periodui.
- Aspiracijos lygis: kriterijai skirti apeiti tabu.

### 2.4.1. NUO LOKALIOS PRIE TABU PAIEŠKOS

Hertz ir kt. [16] publikacijoje tabu paieška įvedama nuo lokalios paieškos, t.y. pagrindinis iteracinės procedūros žingsnis susideda konstruojant iš einamojo sprendinio $i$ kitą sprendinį $j$ ir tikrinant, ar reikia sustoti jame, ar atlikti kitą žingsnį. Aplinkos paieškos metodai yra iteracinės procedūros, kuriuose aplinka $N(i)$ yra apibrėžiama kiekvienam tinkamam sprendiniui $i$, o kitas sprendinys $j$ ieškomas tarp sprendinių aibėje $N(i)$.

Tam, kad pagerinti tyrinėjimo proceso efektyvumą, reikia išaugoti ne tik lokalios informacijos kelią (pavyzdžiui, einamą tikslo funkcijos reikšmę), bet taip pat tam tikrą informaciją susijusią su tyrinėjimo procesu. Sistemingas atminties naudojimas yra esminė tabu paieškos savybė. Nepaisant to, kad dauguma tyrinėjimo metodų saugo iš esmės tik reikšmę $f(i^*)$ geriausio sprendinio $i^*$ aplankyto iki šiol, tabu paieška taip pat saugo informaciją apie paskutinių aplankytų sprendinų maršrutą. Tokia

informacija yra naudinga vadovauti žingsniui nuo $i$ prie kito sprendinio $j$ pasirinktam iš $N(i)$. Atminties vaidmuo yra apriboti pasirinkimą iš tam tikro $N(i)$ poaibio uždraudžiant, pavyzdžui, žingsnius į tam tikrus aplinkos sprendinius.

Tiksliau pastebėsime, kad sprendinio $i$ aplinkos $N(i)$ struktūra bus iš tikrųjų kintama iteracijų metu. Todėl būtų tinkamiau įtraukti tabu paiešką į procedūrų, kurios vadinamos dinaminės aplinkos paieškos metodais, klasę [16].

Formaliai nagrinėkime optimizavimo problemą tokiu būdu: duota galimų sprendinių aibė $S$ ir funkcija $f$: $S \rightarrow \mathbb{R}$, rasti tam tikrą sprendinį $i^*$ iš $S$ tokį, kad $f(i^*)$ yra priimtinas atsižvelgiant į tam tikrą kriterijų (ar kriterijus). Paprastai priimtinumo kriterijus sprendiniui $i^*$ yra $f(i^*) \leq f(i)$ kiekvienam $i$ iš $S$. Tokioje situacijoje tabu paieška būtų tikslus minimizavimo algoritmas arba tyrinėjimo procesas garantuojantis, kad po baigtinio žingsnių skaičiaus toks $i^*$ bus pasiektas.

Daugumoje situacijų nėra garantijos, kad toks $i^*$ bus surastas; todėl tabu paieška paprastai traktuojama kaip ypatingai bendra euristinė procedūra. Kadangi tabu paieška iš tikrųjų įtraukia į savo paties veikimo taisykles kai kuriuos euristinius metodus, tinkamiau nagrinėti tabu paiešką kaip metaeuristiką. Jos vaidmuo dažniausiai yra vadovauti ir orientuoti paiešką iš kitos (labiau) lokalios paieškos procedūros.

Taigi kiekvienas tyrinėjimo procesas turi tam tikrais atvejais priimti taip pat ir nepagerinančius žingsnius nuo $i$ prie $j$ (t.y. $f(j) > f(i)$), jeigu norima išeiti iš lokalaus minimumo.

Kadangi galimi ir nepagerinantys žingsniai, yra rizika aplankyti tą patį sprendinį, t.y. galimi ciklai. Tai yra taškas, kuriame aminties naudojimas yra naudingas uždrausti žingsnius, kurie gali nuvesti prie neseniai aplankytų sprendinių. Jeigu tokia atmintis yra pateikta, galime nagrinėti atvejį, kad $N(i)$ struktūra priklausys nuo maršruto ir taigi nuo iteracijos $k$; taigi galime remtis $N(i, k)$ vietoj $N(i)$.

Svarbu suprasti, jog aibės $N(i, k)$ apibrėžimas kiekvienoje iteracijoje ir jos poaibio $V^*$ pasirinkimas yra kritiniai [16].

$N(i, k)$ apibrėžimas implikuoja tai, jog kai kurie neseniai aplankyti sprendiniai pašalinami iš aibės $N(i)$; jie nagrinėjami kaip tabu sprendiniai, kurių reikia vengti kitoje iteracijoje. Tokia atmintis (paremta naujumu) dalinai apsaugos ciklinimą. Pavyzdžiui, išsaugant iteracijoje $k$ sąrašą $T$ (tabu sąrašą) paskutinių aplankytų $|T|$ sprendinių užkirs kelią daugiausia $|T|$ dydžio ciklams. Tokiu atveju galime paimti $N(i, k) = N(i) - T$.

Dar vienas tabu sąrašo sudarymo trūkumas yra faktas, kad galima situacija, kai suteikiame tabu statusą sprendiniams, kurie gali būti iki tol neaplankyti. Todėl turime suteikti tam tikrą tabu statuso sumažėjimą; tuomet nepaisysime tabu statuso, kada tam tikri tabu sprendiniai atrodo patrauklūs. Tai yra atliekama su *aspiracijos lygio sąlygomis*.

Tabu žingsnis $m$ pritaikytas einamajam sprendiniui $i$ gali pasirodyti patrauklus, nes jis duoda, pavyzdžiui, sprendinį geresnį nei geriausi iki tol rasti. Žingsnį $m$ priimsime nepaisant jo statuso; tai padarysime, jei jis turi *aspiracijos lygį* $a(i, m)$, kuris yra geresnis nei slenksčio reikšmė $A(i, m)$.

Dabar apibūdinome beveik visas pagrindines tabu paieškos sudėtines dalis. Yra dar viena savybė, kuri yra svarbi tyrinėjimo procese; ji susijusi su faktu, kad proceso metu $f$ gali būti pakeista tam tikrais atvejais kita funkcija $\tilde{f}$. Tai leidžia įvesti šiek tiek paieškos intensifikacijos ir diversifikacijos.

Tabu paieškos algoritme atmintis naudojama skirtingais būdais tam, kad vadovautų paieškos procedūrai; jau nagrinėjome trumpalaikę atmintį, kurios vaidmuo buvo uždrausti tam tikrus žingsnius, kurie gali nuvesti atgal prie neseniai aplankytų sprendinių. Atmintis taip pat pateikiama gilesniame lygyje.

Paieškos procese kartais naudinga intensyvinti paiešką tam tikrame aibės $S$ regione, kadangi jame gali būti keletas priimtinų sprendinių. Tokia intensifikacija įgyvendinama suteikiant aukštą prioritetą sprendiniams, kurie turi bendras savybes su einamuoju sprendiniu; tai gali būti atlikta įvedant papildomą narį tikslo funkcijoj; šis narys „baus“ sprendinius, kurie yra toli nuo einamojo sprendinio. Tai turi būti atlikta per keletą iteracijų ir po to naudinga tyrinėti kitą aibės $S$ regioną; diversifikacija tokiu būdu sieks išplėsti tyrinėjimo pastangas virš skirtingų $S$ regionų. Vėlgi diversifikacija gali būti įgyvendinta įvedant papildomą narį tikslo funkcijoj; šis narys baus tam tikroje stadijoje sprendinius, kurie yra arti einamojo sprendinio. Abu intensifikacijos ir diversifikacijos nariai turi svorius, kurie yra keičiami proceso metu tokiu būdu, jog intensifikacijos ir diversifikacijos fazės alternuotų viena su kita paieškos metu. Modifikuota tikslo funkcija yra funkcija $\tilde{f} = f$ + Intensifikacija + Diversifikacija.

Trumpai pristatėme esmines tabu paieškos procedūros charakteristikas, kurios gali būti suformuluotos tokiu būdu [16]:

*Tabu paieška*

1 žingsnis. Pasirinkti pradinį sprendinį $i$ aibėje $S$. Nustatyti $i^* = i$ ir $k = 0$.

2 žingsnis. Nustatyti $k = k+1$ ir sugeneruoti sprendinių poaibį $V^*$ aibėje $N(i, k)$ taip, kad bet kuri iš tabu sąlygų $t_r(i, m) \in T_r$ pažeidžiama ($r = 1, ..., t$) arba bent viena iš aspiracijos sąlygų $a_r(i, m) \in A_r(i, m)$ tenkinama ($r = 1, ..., a$).

3 žingsnis. Pasirinkti geriausią $j = i \oplus m$ aibėje $V^*$ (atsižvelgiant į $f$ ar funkciją $\tilde{f}$) ir nustatyti $i = j$.

4 žingsnis. Jeigu $f(i) < f(i^*)$, tuomet nustatyti $i^* = i$.

5 žingsnis. Atnaujinti tabu ir aspiracijos sąlygas.

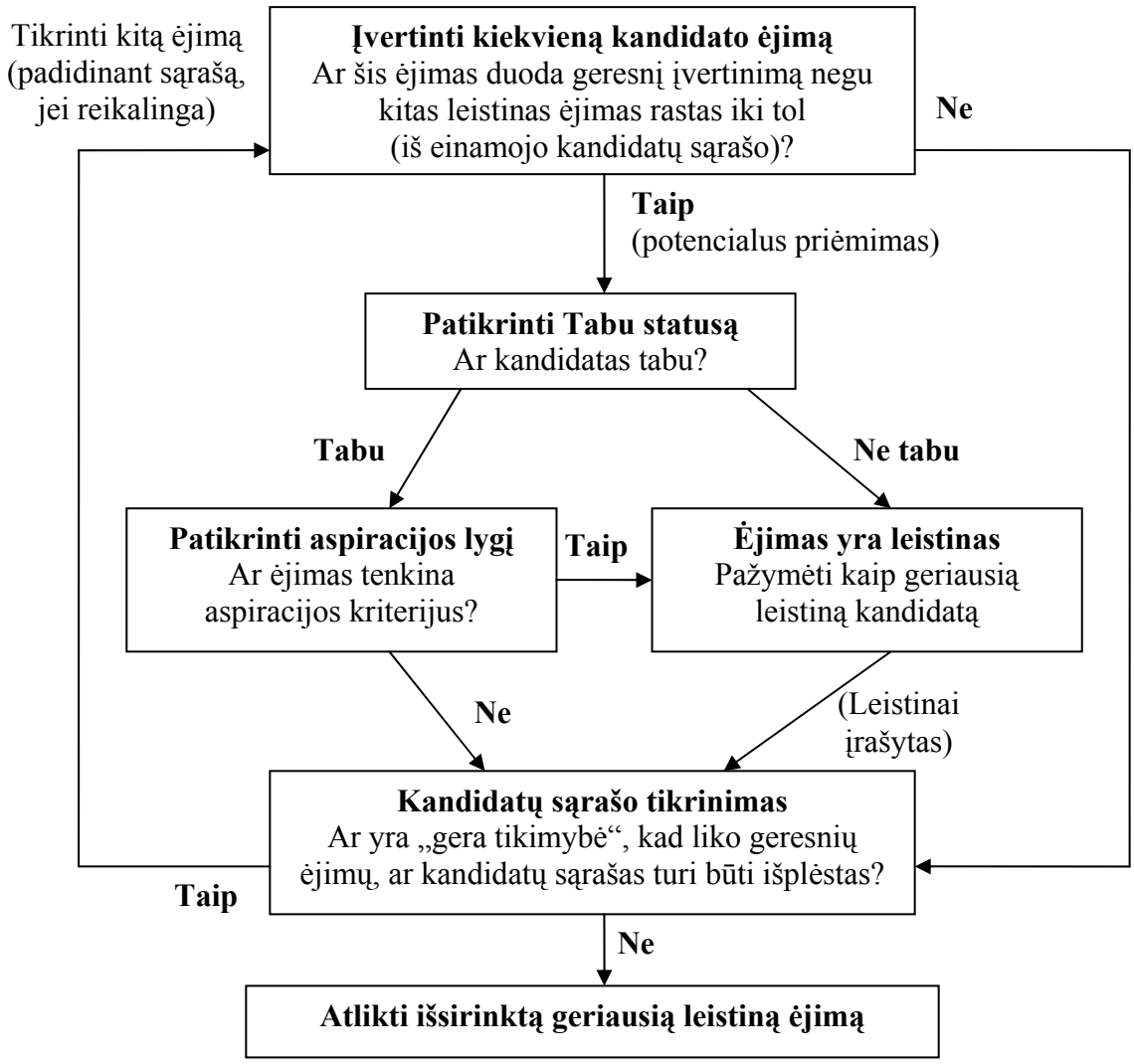
6 žingsnis. Jeigu sustojimo sąlyga tenkinama, tuomet sustoti. Priešingu atveju eiti į 2 žingsnį.

Glover [8] teigė, jog kritinis žingsnis, kuris įgyvendina agresyvią trumpalaikės atminties orientaciją, yra geriausių leistinų kandidatų pasirinkimas (žr. 2.7 pav.).

## 2.4.2. TABU SĄRAŠAS

Tabu sąrašas yra trumpalaikės tabu paieškos atminties komponentas (žr. 2.8 pav.) [8].

Pagrindinis tabu sąrašo vaidmuo yra ciklų prevencija [16]. Jeigu sąrašo ilgis yra per trumpas, šis vaidmuo gali būti nepasiektas; atvirkščiai, per ilgas sąrašas sukuria per daug apribojimų ir pastebėta, kad aplankytų sprendinių vidutinė reikšmė auga kartu su tabu sąrašo dydžio augimu. Paprastai šio dydžio didumo eilė gali būti lengvai determinuota. Tačiau duotai optimizavimo problemai dažnai sunku ar visai neįmanoma rasti reikšmės, kuri apsaugotų nuo ciklų ir ne per daug apribotų paiešką visiems duoto dydžio problemos pavyzdžiams.

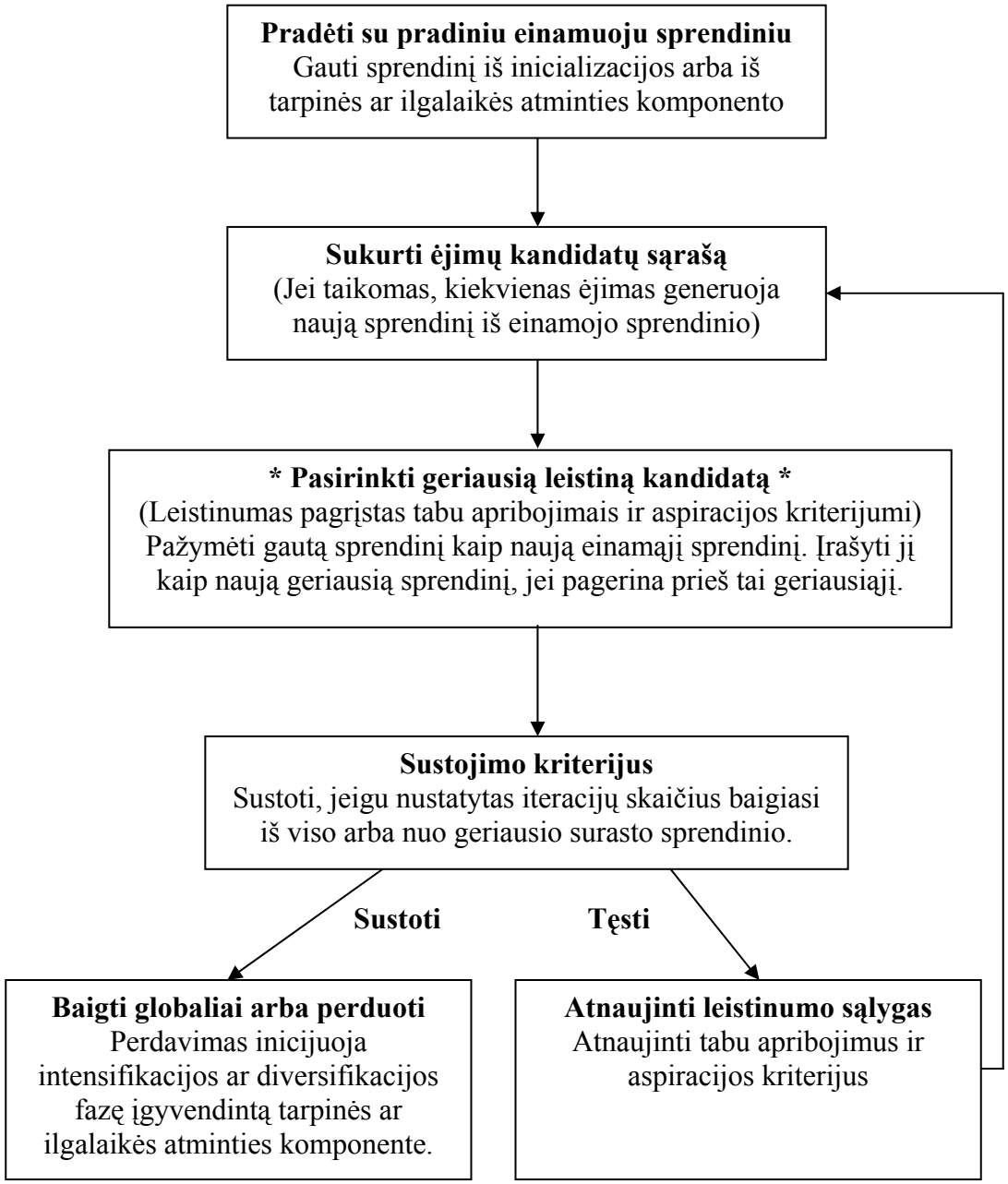



**2.7 pav. Geriausio leistino kandidato išrinkimas [8]**

Efektyvus būdas išvengti šio sunkumo yra naudoti kintamo dydžio tabu sąrašą. Kiekvienas sąrašo elementas priklauso tam tikram iteracijų skaičiui, kuris yra apribotas duotąją maksimalia ir minimalia reikšmėmis.

Tai pat svarbu pastebėti, kad ne visada gera mintis apriboti paieškos erdvę ligi tinkamų sprendinių; daugelyje atvejų, leidžiant paieškai judėti į netinkamus sprendinius yra pageidautina, ir kartais būtina [6].

Glover [8] teigė, kad keliuose ankstesniuose tabu paieškos taikymuose, geriausi tabu sąrašo ilgiai atitinkamai pateko į intervalą nuo 5 iki 12, su 7, reprezentuojančiu efektyviausią reikšmę.




**2.8 pav. Tabu paieškos trumpalaikės atminties komponentas [8]**

Morrison [29] teigė, kad tabu paieškos metodai dažnai naudoja 5-7 žingsnių ilgio tabu atmintį, tačiau optimalus atminties ilgis faktiškai priklauso nuo problemos – didesnei problemai šiek tiek didesnis ilgis.

## 2.4.3. INTENSIFIKACIJA IR DIVERSIFIKACIJA

Du svarbūs tabu paieškos komponentai yra sprendinių intensifikacija ir diversifikacija.

Hertz ir kt. [16] publikacijoje teigiama, jog tam, kad suintensyvintumėme paiešką teikiančiuose vilčių regionuose, iš pradžių turime grįžti prie vieno iš geriausių sprendinių rastų iki tol. Tuomet tabu sąrašas gali būti lengvai sumažintas „mažam“ skaičiui iteracijų.

Kaip minėta prieš tai, dėl priežasčių susijusių su skaičiavimo laiku, greitos euristikos ir priimtino dydžio sprendinio aplinka naudojami kiekviename tabu paieškos žingsnyje. Galimi būdai paieškos intensifikacijai yra sudėtingesnių euristikų ar net tikslių metodų naudojimas, arba sprendinio aplinkos padidinimas.

Tam, kad išvengti didžiulio būsenų erdvės regiono išlikimo visiškai netyrinėtu, svarbu įvairinti paiešką. Paprasčiausias būdas tai padaryti – atlikti keletą atsitiktinių perkrovimų. Kitoks būdas, kuris garantuoja nelankytų vietų eksploraciją yra bausti dažnai atliekamus žingsnius arba sprendinius.

Ši bauda yra nustatoma pakankamai didelė tam, kad užtikrintų pabėgimą iš einamojo regiono. Modifikuota tikslo funkcija naudojama duotam skaičiui iteracijų. Taip pat įmanoma naudoti baudą dažnai atliekamiems žingsniams visos paieškos procedūros metu.

Glover [8] teigė, jog trumpalaikė tabu paieškos atmintis sudaro agresyvios eksploracijos formą, kuri siekia pagaminti geriausią (aukščiausio įvertinimo) ėjimo galimybę su sąlyga, kad galimi pasirinkimai tenkintų tam tikras sąlygas. Šios sąlygos, įgyvendintos į tabu apribojimus, yra kuriamos apsaugoti pakeitimus, ar kartais pasikartojimus, tam tikrų ėjimų – pateikiant pasirinktus atributus šių ėjimų draudžiamais (tabu). Pirminis tabu apribojimų tikslas yra leisti metodui eiti virš lokalaus optimalumo taškų, tačiau nepaisant to gaminant aukštos kokybės ėjimus kiekviename žingsnyje.

[6] šaltinyje teigiama, kad intensifikacija naudojama daugelyje tabu paieškos įgyvendinimų, bet ne visada yra būtina. Tai yra dėl to, jog yra daug situacijų, kuriose paieškos, atliktos normaliu paieškos procesu, pilnai pakanka. Tokiu būdu nereikia eikvoti laiko tyrinėjant atidžiau paieškos erdvės dalis, kurios jau buvo aplankytos, ir šis laikas gali būti panaudotas daug efektyviau.

Diversifikacija yra algoritminis mechanizmas, kuris stengiasi sumažinti lokalumo problemą priversdamas paiešką judėti po anksčiau netyrinėtus paieškos erdvės plotus.

### 2.4.4. ALGORITMO ĮGYVENDINIMO ASPEKTAI

Hertz ir kt. [16] teigė, kad iteracinių sprendinių metodų efektyvumas dažniausiai priklauso nuo modeliavimo. Tikslus parametrų suderinimas niekada nenusvers blogo pasirinkimo aplinkos struktūros arba tikslo funkcijos. Žvelgiant iš kitos pusės, efektyvus modeliavimas privalo vesti prie robastinių technikų, kurios nėra per jautrios skirtingiems parametrų nustatymams.

Nors iš esmės paprastas metodas yra skolingas savo efektyvumą pakankamai geram suderinimui aiškiai didelio parametrų rinkinio.

Dabartiniu momentu, galime pripažinti, kad šis technika yra panašesnė į inžinerinį metodą sudėtingoms ir didžiulio formato optimizavimo problemoms nei į elegantišką paprastą matematinį algoritmą [16].

Naudojant tabu paiešką, sudėtingumas yra ne tik sprendžiamose problemose, bet ir pačios sprendimo technikos pritaikyme.

Gendreau [6] pateikė tokius algoritmo sustabdymo kriterijus

- po fiksuoto iteracijų skaičiaus (arba fiksuoto procesoriaus laiko kiekio);
- po tam tikro skaičiaus iteracijų be pagerėjimo tikslo funkcijos reikšmėj (kriterijus naudojamas daugelyje įgyvendinimų);
- kuomet tikslo funkcija pasiekia iš anksto nustatytą slenksčio reikšmę.

Sudėtingose tabu schemose, paieška paprastai sustoja po fazių sekos baigimo, kiekvienos fazės trukmė nustatoma minėtais kriterijais.

Gendreau [6] publikacijoje ypač akcentuojamas diversifikacijos vaidmuo: „Prieš užbaigiant (...), norėtumėme pabrėžti, kad tinkamos paieškos diversifikacijos užtikrinimas yra galbūt lemiamas klausimas tabu paieškos euristikų projektavime.“

# 3. TIRIAMOJI DALIS IR REZULTATAI

Vienas iš šio darbo tikslų buvo pritaikyti specifinius problemai genetinį, tabu, atsitiktinio klaidžiojimo ir tikslios paieškos algoritmus, tačiau išlaikančius visas bendrųjų algoritmų (ypač genetinio ir tabu) savybes 1.2 skyrelyje minėtoms gamybinių tvarkaraščių sudarymo problemoms spręsti. Įgyvendintos programos tikslai gali būti apibrėžiami tokiu būdu:

- interaktyvi vartotojo sąsaja;
- skirtingų duomenų formatų nuskaitymas ir įvedimas atitinkamai pagal skirtingą sprendžiamą tvarkaraščių sudarymo problemą;
- problemos sudėtingumo pateikimas tikslios (arba pilnosios paieškos) ir atsitiktinės paieškos algoritmų pagalba;
- optimalaus uždavinio sprendinio radimas genetinio ir tabu paieškos algoritmų pagalba;
- rezultatų tekstinis ir grafinis pateikimas.

## 3.1. BENDRŲ ALGORITMO DALIŲ ĮGYVENDINIMAS

Problemos ir jos sprendimo vientisumui užtikrinti visiems pritaikytiems algoritmams naudojamas tas pats permutacinis sprendinio kodavimas bei sprendinio įverčio funkcija.

### 3.1.1. DUOMENŲ KODAVIMAS

Iš pradžių turime pradinių duomenų matricą $A$, susidedančią iš $m$ eilučių ir $n$ stulpelių, kur eilutės žymi atitinkamas mašinas, stulpeliai darbus, o matricos elementai $A_{i,j}$ $j$-tojo darbo trukmę $i$-tojoje mašinoje. Bei to paties formato apribojimų matricą $B$, kurioje yra darbų atlikimo atitinkamose mašinose tvarka.

Sukuriame sprendinio reprezentaciją $X$, kuri yra matrica iš 2 eilučių ir $c$ stulpelių, kur $c = n \cdot m$. Į visus jos stulpelius surašome visus įmanomus matricos $A$ indeksų kėlinius, kurių, akivaizdu, šiuo atveju yra $n \cdot m$. Vadinasi, tokiu sprendinio kodavimu gauname sprendinio reprezentaciją, kuri atitinka darbų eiliškumo seką, t.y., pavyzdžiui, jei chromosomos $k$-tajo stulpelio pirmoje eilutė yra reikšmė 3, o antroje 2, vadinasi, $k$-toje eilėje reikia atlikti 2-ą darbą 3-ioje mašinoje. Kaip jau minėjome, toks kodavimas vadinamas permutaciniu arba kėlinių kodavimu. Akivaizdu, jog šiuo atveju yra $c!$ visų galimų kėlinių. Vadinasi, tiek elementų sudaro sprendinio paieškos erdvę.

### 3.1.2. ĮVERČIO FUNKCIJA

Sukuriame matricą-stulpelį $Pf$, kuris susideda iš $p$ eilučių. Kiekvieną iš eilučių užpildome tokiu būdu $Pf_i = f(P_i)$, kur $1 \le i \le p$. T.y. matrica $Pf$ yra visų (populiacijos ar lokalios aplinkos)

sprendinio reprezentacijų įverčių matrica. Tam sukūrėme ir įgyvendinome įverčio funkciją $f(X)$, kuri įvertina kiekvieną reprezentaciją $X$ toliau aprašytuoju būdu.

Papildomai suformuokime matricas stulpelius $Tr$ iš $m$ eilučių, kuriame bus fiksuojamas $i$-tosios mašinos darbo pabaigos laikas duotuoju momentu ($1 \le i \le m$), ir $D$ iš $n$ eilučių, kuriame bus fiksuojamas $j$-tojo darbo pabaigos laikas duotuoju momentu ($1 \le j \le n$). Pradiniu laiko momentu abiejų matricų elementai yra *0*. Iš eilės tikriname nagrinėjamos chromosomos stulpelius $k$ ($1 \le k \le c$). Pagal chromosomos užkodavimą, *1*-a jos eilutė yra mašinos numeris, o *2*-oji – darbo numeris, vadinasi $k$-tajame tikrinime gausime dvi reikšmes:

$$e = X_{0,\,k} \text{ ir } s = X_{1,\,k},$$

jas įsistačius į matricą $A$, gauname $s$-tojo darbo trukmę $e$-tojoje mašinoje, t.y. $t = A_{e,s}$. Tuomet gauname tokias sąlygas

$$\begin{cases} Tr_e = D_s + t, & jei \;\; Tr_e < D_s \\ Tr_e = Tr_e + t, & jei \;\; Tr_e \ge D_s \end{cases},$$

bei $D_s = Tr_e$. Jas galima paaiškinti taip: jei nagrinėjamas $s$-tasis darbas tuo metu dar vykdomas kurioje nors kitoje mašinoje, vadinasi, reikia palaukti jo pabaigos ir tik tada pradėti vykdyti $e$-tojoje mašinoje, priešingu atveju $s$-tasis darbas iškart vykdomas $e$-tojoje mašinoje.

Tačiau toks chromosomos įvertinimas yra galimas tik atvirojo fabriko problemoms, darbų ir srautinio fabriko problemoms reikia papildomai įvesti matricą-stulpelį $Se$ iš $n$ eilučių, kuriame būtų saugomi mašinų numeriai, kuriose atitinkamas darbas dar nebuvo, vadinasi prie matricos $A$ $e$-tosios eilutės ir $s$-tojo stulpelio grąžinamos reikšmės, reikėtų grąžinti ir apribojimų matricos $B$ atitinkamą reikšmę $eil$, ir prieš tikrinant aukščiau paminėtas sąlygas, reikėtų tikrinti sąlygą $eil = Se_s$ kuri reikštų, ar šis darbas gali būti vykdomas šioje mašinoje, t.y. ar laikomasi mašinų eiliškumo tvarkos.

Galiausiai abiem atvejais chromosomos įvertis yra $\max\limits_{0 \le i \le m}(Tr_i)$, kas atitinka vėliausiai pasibaigusį darbą $i$-tojoje mašinoje.

Pastebėsime, kad visi tvarkaraščių sudarymo apribojimai yra iškart įtraukiami į įverčio funkcijos reikšmę, t.y. tokiu būdu galime naudoti visus sprendinių erdvės elementus, nes šiuo atveju visi jie yra tinkami sprendiniai (su geresne ar blogesne, priklausomai nuo apribojimų, įverčio funkcijos reikšme).

## 3.2. GENETINIO ALGORITMO TAIKYMAS

Genetinio algoritmo taikymo aprašymui naudosime schemą, aprašytąją 2.3.1. skyrelyje, konkrečiai pritaikytą mūsų atvejui.

### 3.2.1. POPULIACIJOS SUDARYMAS

Pradinės populiacijos inicializacijai sugalvojome paprastą, tačiau pakankamai efektyvų elementų generavimo metodą. Suformuojame matricą-stulpelį $P$ iš $p$ eilučių. Jį užpildome chromosomomis $X$, kurios yra atsitiktinai suformuotos iš pradinės matricos $X$, t.y.

1) iš pradžių nukopijuojame chromosomą $X$;

2) pradedame nagrinėti visus jos stulpelius $X_j$ nuo pradžių ($j=1$), ties kiekvienu stulpeliu sugeneruojame atsitiktinį sveikąjį skaičių $rr \in [1;c]$ (tolygus intervale [1; c] skirstinys), ir sukeičiame $j$-tąjį stulpelį su $rr$-tuoju, tada einame prie sekančio stulpelio ir taip iki kol prieisime paskutinį.

3) Gauname visiškai skirtingą nuo pradinės chromosomą $X'$, ją įtraukiame į populiaciją. Ir naujai chromosomai gauti kartojame 1) žingsnį iki kol neužpildoma visa populiacija.

Tokiu būdų sugeneruota chromosomų populiaciją bus visiškai atsitiktinė ir gerai padengs sprendimo erdvę.

### 3.2.2. NAUJOS POPULIACIJOS SUDARYMAS

Sukuriame naują tokio paties dydžio kaip ir $P$ populiaciją $P'$, kartodami tokius žingsnius, kol neužpildoma visa populiacija.

**1) [Atranka]**

Populiacijų atrankai įgyvendinti naudojame sugalvotą ruletės rato atrankos metodo modifikaciją. Iš pradžių apskaičiuojama reikšmės

$$l_{\max} = \max_{1 \le i \le p}(Pf_i),$$

$$\max = l_{\max} + l_{\max} \cdot 0.1,$$

$$suma = \sum_{i=1}^{p}(\max - Pf_i),$$

$$koef = 100 / suma,$$

tada sugeneruojamas atsitiktinis realus skaičius $r$ iš intervalo $[0; 100)$ ir tikrinama sąlyga

$$\begin{cases} nr1 = j, \text{ jei } \sum_{i=1}^{j}(\max - Pf_i) \cdot koef \ge r \\ j = j + 1, \text{ priešingu atveju} \end{cases}, \text{ čia } j = \overline{1,\ldots,p}.$$

Taip gauname pirmąją chromosomą-tėvą, kurios numeris $nr1$. Po to panaši procedūrą kartojama, tik papildomai tikrinama sąlyga, kad antroji chromosoma-tėvas, kurios numeris $nr2$ tenkintų sąlygą $nr2 \ne nr1$. Taip pat kryžminimo procedūrai perduodame kryžminimo numerį $k$. Šios modifikacijos esmė ta, kad net ir blogiausią įvertį turinti chromosoma turi šansą (nors ir nedidelį) pakliūti į atranką.

**2) [Kryžminimas]**

Po atrankos gavome dvi populiacijos $P$ chromosomas $P_{nr1}$ ir $P_{nr2}$. Sugeneruojame du atsitiktinius skaičius $r$ ir $rr$, sveikąjį ir realų, atitinkamai iš intervalų $[1; c-1)$ ir $[0; 1)$. Tikriname, ar $rr \le \rho_K$, jei taip atliekame chromosomų kryžminimą, t.y. iš $P_{nr1}$ chromosomos paimame $r$ pirmų stulpelių ir nukopijuojame į naująją chromosomą $P'_{k+1}$, atitinkamai iš chromosomos $P_{nr2}$ į $P'_k$. Likusioji dalis užpildoma tradiciniu (tiesioginiu) būdu – tikriname nuo pradžios chromosomos $P_{nr2}$ stulpelius ir žiūrime, ar tokių pačių nėra $P'_{k+1}$ chromosomoj, jei taip, tai nukopijuojame atitinkamą stulpelį į chromosomą $P'_{k+1}$. Taip pat ir su $P_{nr1}$ ir $P'_k$ chromosomomis.

Jei kryžminti nereikia, tai chromosomos-palikuonys $P'_k$ ir $P'_{k+1}$ yra tikslios chromosomų-tėvų atitinkamai $P_{nr1}$ ir $P_{nr2}$ kopijos.

**3) [Mutacija]**

Chromosomos mutacijai atlikti, iš pradžių, sugeneruojame du atsitiktinius sveikuosius skaičius $p1$ ir $p2$ iš intervalo $[1; c]$ bei vieną realų $r$ iš intervalo $[0; 1)$. Tikriname, ar $r \le \rho_M$, jei taip, vadinasi, sukeičiame mutuojančios chromosomos stulpelius, kurių indeksai $p1$ ir $p2$ vietomis.

## 3.2.3. ALGORITMO PABAIGA

Senoji populiacija $P$ naująja populiacija $P'$ pakeičiama chromosomų atrankos metodo pabaigoje. T.y. iš pradžių atrenkamos chromosomos kryžminimui, jos sukryžminamos ir įtraukiamos į naująją populiaciją $P'$. Iš viso atliekama $(p - Es)/2$ kryžminimų. Tada $p - Es$ populiacijos $P'$ chromosomų mutuoja/nemutuoja, o likusioji dalis užpildoma geriausiomis chromosomomis iš $P$, neatliekant jokių pakeitimų, t.y. tomis chromosomomis, kurių įvertis yra mažiausias.

Bazinio algoritmo pabaigos sąlyga yra apibrėžtas iteracijų skaičius *IterSk*.

## 3.3. TABU PAIEŠKOS ALGORITMO TAIKYMAS

Įgyvendintas tabu paieškos algoritmas remiasi 2.4.1 skyrelyje minėtomis idėjomis ir bendrąja algoritmo schema.

## 3.3.1. INICIALIZACIJA

Kadangi naudojamas tas pats sprendinių kodavimas kaip ir genetiniame algoritme, galime naudoti ir panašius žymėjimus tabu paieškos algoritmo įgyvendinimui. Iš pradžių sukuriama *c* ilgio matrica-eilutė $X_{\text{TM}}$ globaliam minimumui saugoti, kurį užpildome tokiu pačiu atsitiktiniu būdu kaip ir genetiniame algoritme pildomas populiacijos elementas. Toliau sukuriamos dar dvi matricos. Pirmoji

$c \times ck$ formato TK matrica – sprendinio aplinkos sprendinių saugojimui, kur $ck$ aplinkos sprendinių dydis. Matricos TK pirmasis elementas yra einamasis sprendinys, visa likusi matricos dalis užpildoma aplinkos sudarymo funkcijos pagalba (žr. kitą skyrelį). Antroji $c \times ct$ formato matrica TS skirta tabu sąrašui, t.y. tabu sprendiniams saugoti, čia $ct$ yra tabu sąrašo ilgis. Iš pradžių ši matrica užpildoma pradiniu sprendiniu, nes tuo metu tik jis yra tabu.

### 3.3.2. APLINKOS SUDARYMO FUNKCIJA

Aplinkos sudarymo funkcijos $f$ principas paremtas Hemingo atstumo (žr. [13]) tarp dviejų simbolių eilučių idėja. Kitais žodžiais tariant, norint neišeiti iš lokalios aplinkos apibrėžimo rėmų, einamajam sprendiniui turime sudaryti vienos transformacijos taikymo sprendiniui aplinką, arba aplinką, kurios narių Hemingo atstumas iki einamojo sprendinio yra lygus 2. Vadinasi,

$f(k, j): X \rightarrow X'$,

kur $X=\{x_1, \ldots, x_k, \ldots, x_i, \ldots, x_c\}$, $X'=\{x_1, \ldots, x_i, \ldots, x_k, \ldots, x_c\}$, kur $i>k$, $i=2, ..., c$; $k=1, ..., c$-1. Čia indeksas $k$ reprezentuoja stabiliosios pozicijos elementą, o indeksas $i$ keitimo pozicijos elementą, t.y. kintant $i$, $k$ lieka pastovus. Kadangi bendru atveju $ck \geq c$, tuomet kai keitimo pozicijos $i$ reikšmė tampa lygi $c$, stabilioji pozicija $k$ padidinama vienetu, o $i$ priskiriama $k$+1 reikšmė.

Aplinkos funkcija taikoma $ck$-1 kartų. Nesunku įsitikinti, kad tokiu būdu be pasikartojimų galima užpildyti $\sum_{n=1}^{c-1} n$ galimų skirtingų kėlinių, kurių Hemingo atstumas nuo bazinio kėlinio bus lygus 2, ko pilnai pakanka sprendinio aplinkos sudarymui. 3.1 lentelėj pateikiamas aplinkos sudarymo funkcijos veikimo pavyzdys, kuomet turime $n$=4 ilgio kėlinį, viso galimų kėlinių $n!$=24. Kaip matome, po 6 (t.y. 1+2+3) žingsnių algoritmas stabdomas.

**3.1 lentelė**

**Aplinkos sudarymo funkcijos veikimo pavyzdys**

| Algoritmo žingsnis | Stabilioji pozicija *k* | Einamoji pozicija *i* | Kėlinys *j* žingsnyje |
|---|---|---|---|
| 0 | - | - | 1234 |
| 1 | 1 | 2 | 2134 |
| 2 | 1 | 3 | 3214 |
| 3 | 1 | 4 | 4231 |
| 4 | 2 | 3 | 1324 |
| 5 | 2 | 4 | 1432 |
| 6 | 3 | 4 | 1243 |

### 3.3.3. GERIAUSIO KANDIDATO IŠRINKIMAS IR ATSITIKTINIAI STARTAI

Norint atlikti tolimesnį algoritmo žingsnį, reikia įvertinti visus sprendinius-kandidatus, t.y. sprendinio aplinkos narius. Tam iš pradžių įvertinamas pats globalus (čia globalus algoritmo paieškos prasme) sprendinys $\min_0=f(X_{TM})$. Toliau įvertinami visi einamojo sprendinio aplinkos nariai, t.y. $\min_i=f(TK_i)$, $i=1, ..., ck$. Tuomet iš gautų įverčių išrenkamas geriausias. Jis lyginimas su tabu sąrašo TS elementais ir galimi tokie atvejai:

- jei elemento nėra tabu sąraše ir jis mažesnis už surastą globalų minimumą $X_{TM}$, tai globalus ir einamasis minimumai pakeičiami šiuo elementu – kandidatas išrinktas;
- jei elemento nėra tabu sąraše, tačiau jis didesnis už globalų minimumą, einamasis minimumas pakeičiamas šiuo elementu – kandidatas išrinktas;
- jei elementas yra tabu sąraše, tačiau jis mažesnis už globalų minimumą, globalus ir einamieji minimumai pakeičiami šiuo elementu – kandidatas išrinktas;
- jei pasiektas paskutinis sprendinio aplinkos narys, einamasis minimumas juo pakeičiamas – kandidatas išrinktas;
- visais kitais atvejais kandidato paieška tęsiama iki kol netenkinamas bent vienas iš prieš tai minėtų sąlygų.

Paskutinis ir labai svarbus tabu paieškos aspektas yra atsitiktiniai startai, t.y. algoritmo eigoj, kuomet geriausio surasto minimumo reikšmė nepagerinama $m_{iter}$ kartų, kur $m_{iter} = iter \cdot k_{rand}$, čia $iter$ – algoritmo iteracijų skaičius, $k_{rand}$ – atsitiktinių startų koeficientas (šiuo atveju pasirinkta 0.2 reikšmė), einamasis sprendinys sugeneruojamas atsitiktinai (tabu sąrašas neišvalomas) ir algoritmas tęsiamas nuo šio sprendinio.

## 3.4. PILNOS IR ATSITIKTINĖS PAIEŠKOS ALGORITMŲ TAIKYMAS

Norėdami pavaizduoti nagrinėjamos problemos sudėtingumą pritaikėme du įgyvendinimo prasme paprastus algoritmus: tikslios (pilnutinės) paieškos ir atsitiktinio klaidžiojimo.

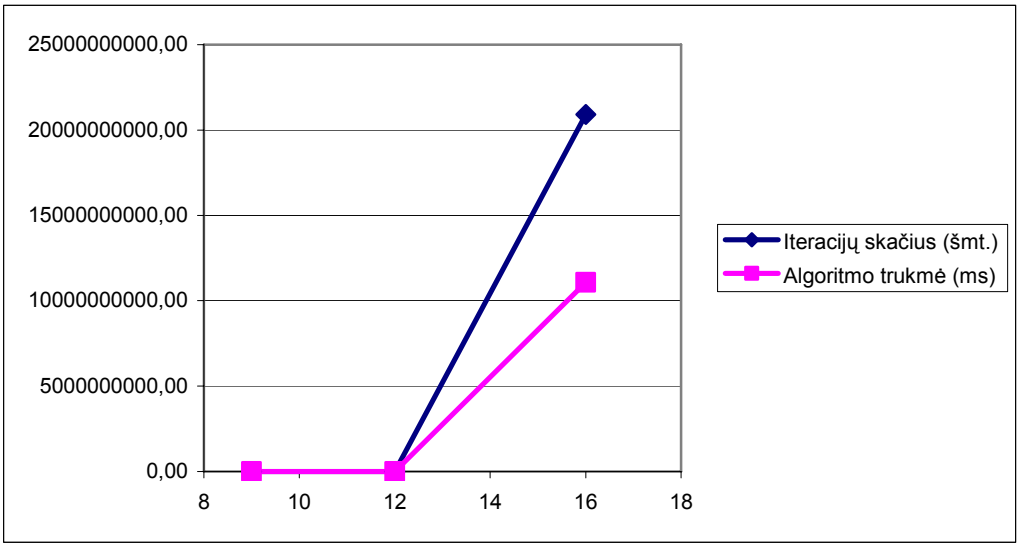
Kaip jau minėjome permutacinio sprendinio kodavimo atveju visų galimų kėlinių (sprendinių) skaičius yra lygus $n!$. Taigi reikia nagrinėti $n!$ sprendinių norint surasti tikslią globalaus optimumo reikšmę. Šiam tikslui naudotas visų galimų kėlinių generavimo algoritmas, kurį galima rasti [36] šaltinyje. Algoritmo sustojimo kriterijus yra pilnas (lygus $n!$) arba dalinis (mažesnis už $n!$) iteracijų skaičius, arba nustatyta algoritmo trukmė.

Kaip pilnosios paieškos pavyzdį, galime pateikti vieną iš atvirojo fabriko problemų ($3 \times 3$ formato), kuriai pabandėme rasti tikslų sprendinį. Programa jį apskaičiavo per 1s 92ms. Atlikę paprastus algebrinius skaičiavimus, radome kiek apytikriai laiko programa skaičiuotų šiek didesnio formato problemas. Rezultatai pateikti 3.2 lentelėje ir 3.1 pav.

**3.2 lentelė**

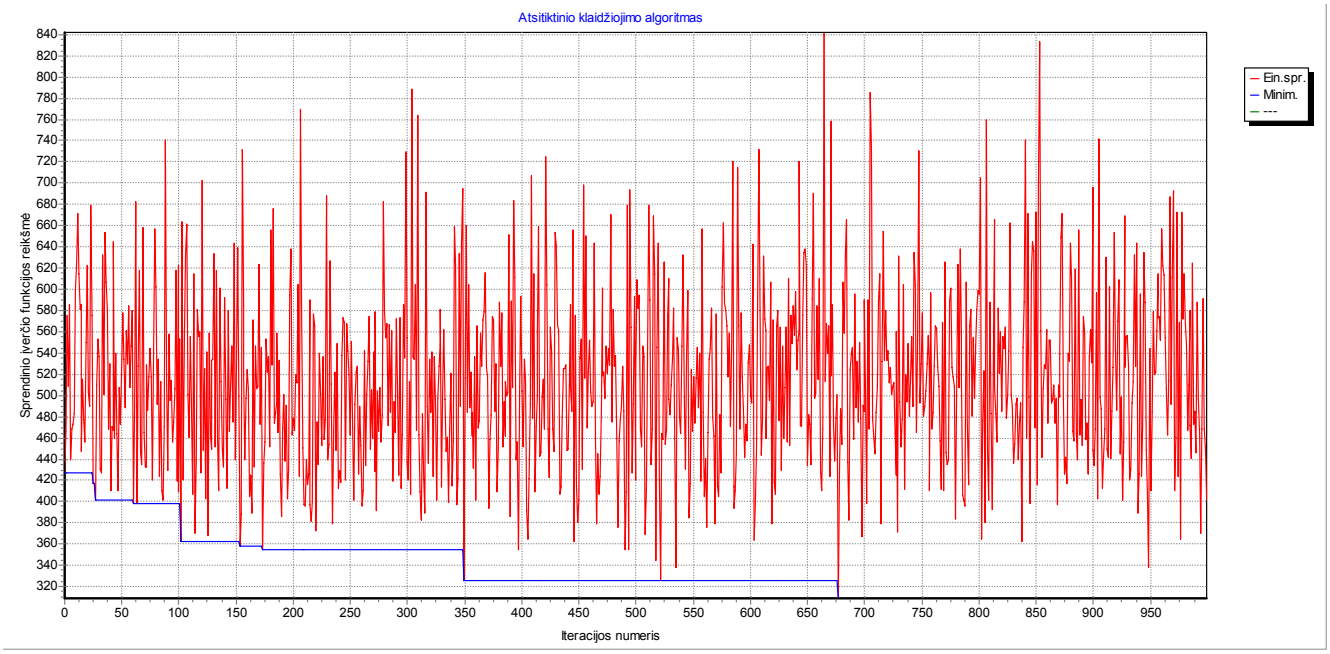
**Tikslaus (pilnos paieškos) algoritmo taikymo pavyzdys**

| Problemos formatas (kodavimo ilgis) | Iteracijų (sprendinių) skaičius | Optimalių sprendinių skaičius | Algoritmo trukmė |
|---|---|---|---|
| 3 × 3 (9) | 362880 | 3395 | 1s 92ms |
| 3 × 4 (12) | 479001600 | --- | ≈42min 14s 40 ms |
| 4 x 4 (16) | 2092289888000 | --- | ≈3 metai 6mėn. 3 paros 6h 43min 12s 00ms |




**3.1 pav. Pilnosios paieškos algoritmo iteracijų skaičius (šimtais iteracijų) ir trukmė (milisekundėmis) atvirojo fabriko 3 × 3, 3 × 4 ir 4 × 4 formato problemoms**

Kitam pritaikytam algoritmui - atsitiktiniam klaidžiojimui – naudota paprasta strategija (apie panašaus pobūdžio algoritmus žr. [19]): iš pradžių sukuriamos dvi $c$ ilgio matricos-eilutės $X_M$ ir $X_E$, atitinkamai geriausiam surastam sprendiniui ir einamajam sprendiniui saugoti. $X_E$ užpildome tokiu pačiu atsitiktiniu būdu kaip ir genetiniame bei tabu paieškos algoritmuose. Atlikę vieną žingsnį, tikriname, ar suradome geresnį sprendinį, jei taip, jį išsaugome, jei ne, sugeneruojame naują elementą. Algoritmo sustojimo kriterijus yra nustatytas iteracijų skaičius, arba nustatyta algoritmo trukmė. Algoritmo pavyzdys pateiktas 3.2 pav.




**3.2 pav. Atsitiktinio klaidžiojimo algoritmo atvirojo fabriko (4 x 4 formato, 1000 algoritmo iteracijų) problemos sprendinių kitimo grafikas**

## 3.5. TYRIMAI

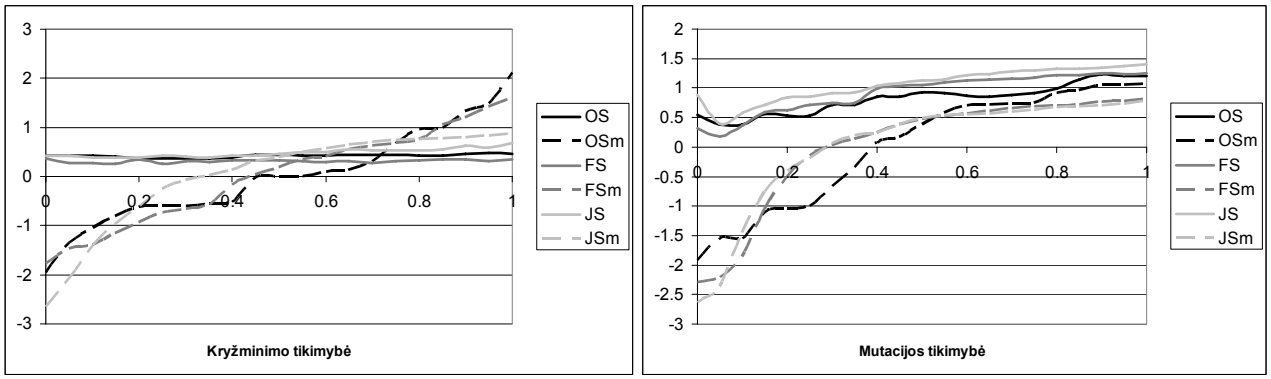
Duomenys eksperimentams paimti iš Vakarų Šveicarijos taikomųjų mokslų universiteto profesoriaus E. Taillard puslapio [37].

### 3.5.1. GENETINIO ALGORITMO PARAMETRŲ ĮTAKOS TYRIMAS

Kiekvienai problemai $r$=100 kartų leidžiamas genetinis algoritmas su atitinkamais parametrų nustatymais, kurie apima visą galimą arba beveik visą galimą reikšmių intervalą. Rezultate išvedami apskaičiuotų populiacijų chromosomų įverčių minimumų, vidurkių standartizuotos reikšmės. Chromosomų įverčių standartiniam nuokrypiui skaičiuoti taikėme patogią skaičiavimams formulę [4]: $s = \sqrt{\left(x_1^2 + x_2^2 + ... + x_n^2\right)/\left(n-1\right) - \left(n \cdot \overline{x}^2\right)/\left(n-1\right)}$, o standartizuotoms reikšmėms: $z_i = \left(x_i - \overline{x}\right)/s$.

Algoritmo pradiniai duomenys: iteracijų skaičius *IterSk* = 5 · *P*, populiacijos dydis *P* = *m* · *n* (t.y. lygus sprendinio kodavimo ilgiui), chromosomų kryžminimo tikimybė *K* = 0.8, chromosomų mutacijos tikimybė *M* = 0.1, populiacijos elitarizmo procentas *E* = 20%. Keičiant tiriamąjį parametrą kiti parametrai lieka stabilūs.

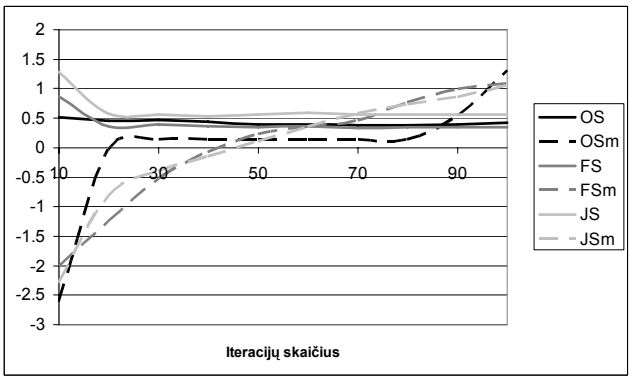
Grafiniai rezultatai pateikti 3.2 paveiksle. Juose vientisa kreivė vaizduoja standartizuotų populiacijų sprendinių vidurkį, o punktyrinė – standartizuotus populiacijų minimumus, kintant atitinkamam parametrui. Juodos spalvos kreivė – atvirajam fabrikui (OS), pilkos – srautiniam (FS), šviesiai pilkos – darbų fabrikui (JS), jų minimumams atitinkamai OSm, FSm, JSm aprašo kreivės.



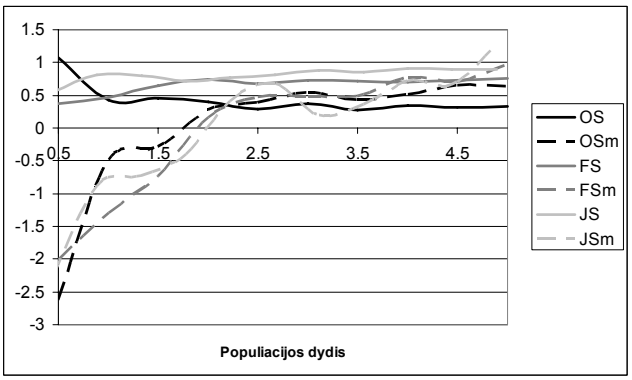

**(a)**



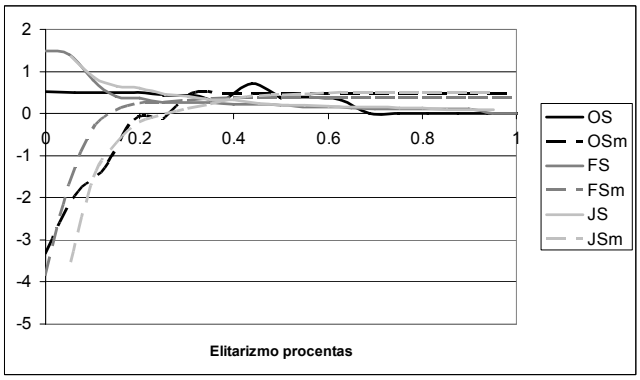

**(b)**




**(c)**




**(d)**



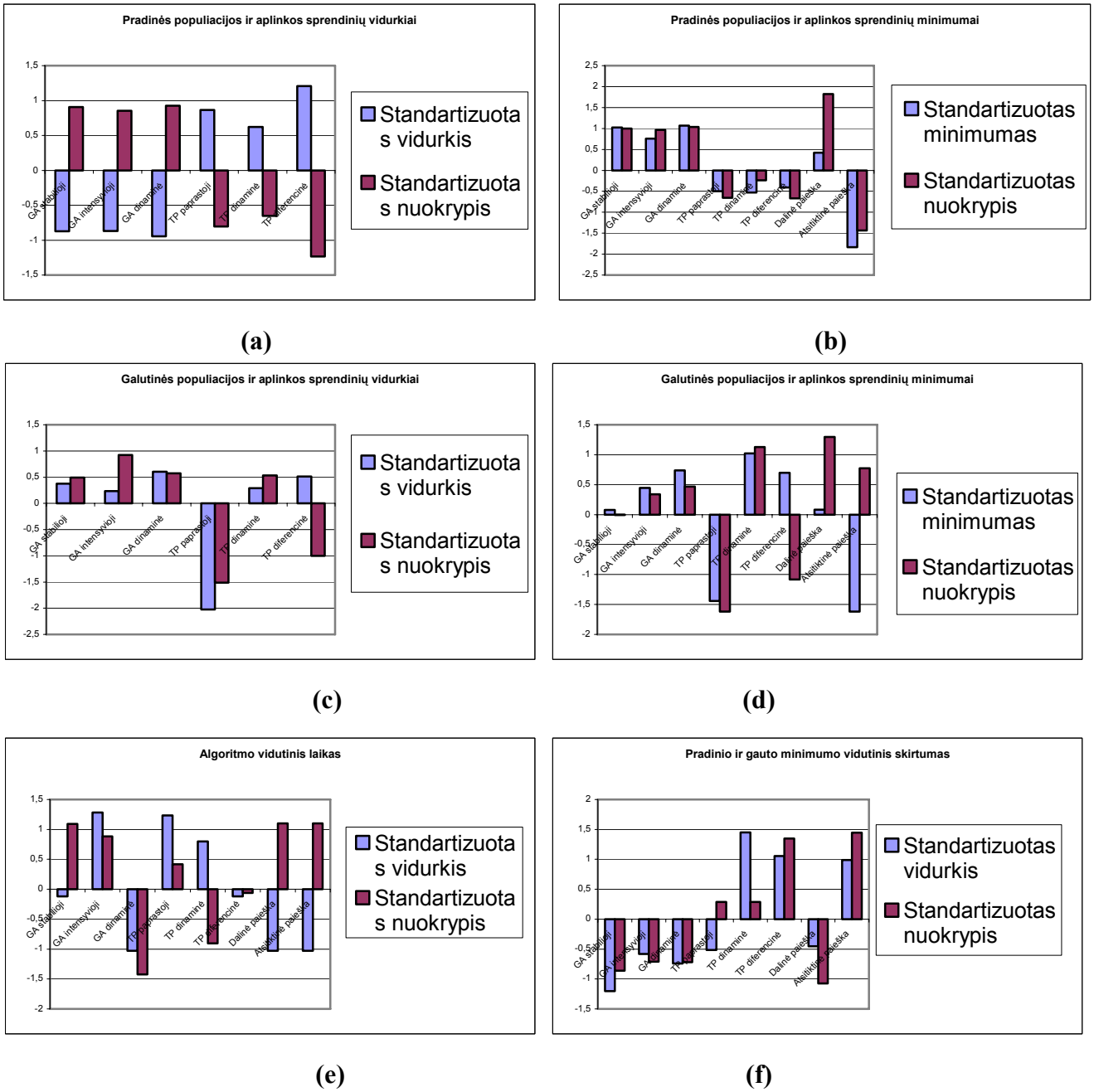

**(e)**

**3.3 pav. Genetinio algoritmo parametrų kitimo įtaka: a) kryžminimo tikimybės parametro; b) mutacijos tikimybės parametro; c) iteracijų skaičiaus parametro; d) populiacijos dydžio skaičiaus parametro įtaka; e) elitarizmo procento parametro.**

## 3.5.2. ALGORITMŲ EFEKTYVUMO TYRIMAS

Algoritmų efektyvumui kiekvienai tvarkaraščių sudarymo klasei sugeneravome 100 atsitiktinių 10 x 10 formato problemų ir 100 kartų taikėme genetinį algoritmą su stabiliąja, intensyviąja bei dinamine, tabu paieškos algoritmą su paprastąja ir padidintos diversifikacijos strategijomis, dalinės bei atsitiktinės paieškos algoritmus. Visi algoritmai buvo leidžiami tam tikrą iteracijų skaičių, tik dalinės

bei atsitiktinės paieškos algoritmai buvo tiek, kiek ilgiausiai laiko užtruko prieš tai leistas algoritmas. Gauti rezultatai pateikti standartizuotoje (kokybinėje) formoje 3.4A ir 3.4B pav.




**3.4 pav. Septynių algoritmų efektyvumo tyrimo kokybiniai grafikai**

# 4. PROGRAMINĖ REALIZACIJA IR VARTOTOJO INSTRUKCIJA

Šiame darbe nagrinėjamoms tvarkaraščių sudarymo problemoms spręsti bei algoritmams realizuoti „Borland C++ Builder“ terpėje (remiantis [1], [2]) buvo sukurta programinė įranga „GATS“ (programinė įranga, naudoti duomenys, tarpiniai skaičiavimai bei visi rezultatai yra pridėtame kompaktiniame diske).

Programos paleidimo failas yra „GATS.exe“.

## 4.1. DARBAS SU PROGRAMA

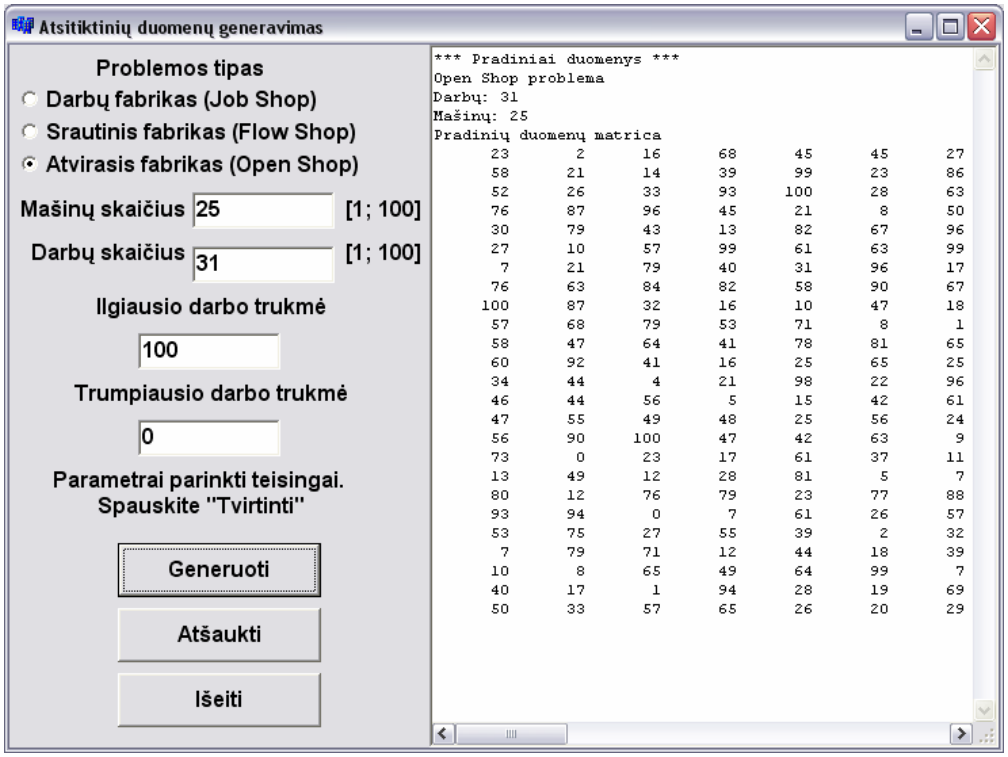
Programos meniu punktai bei jų paskirtis pavaizduoti 4.1 lentelėje.

**4.1 lentelė**

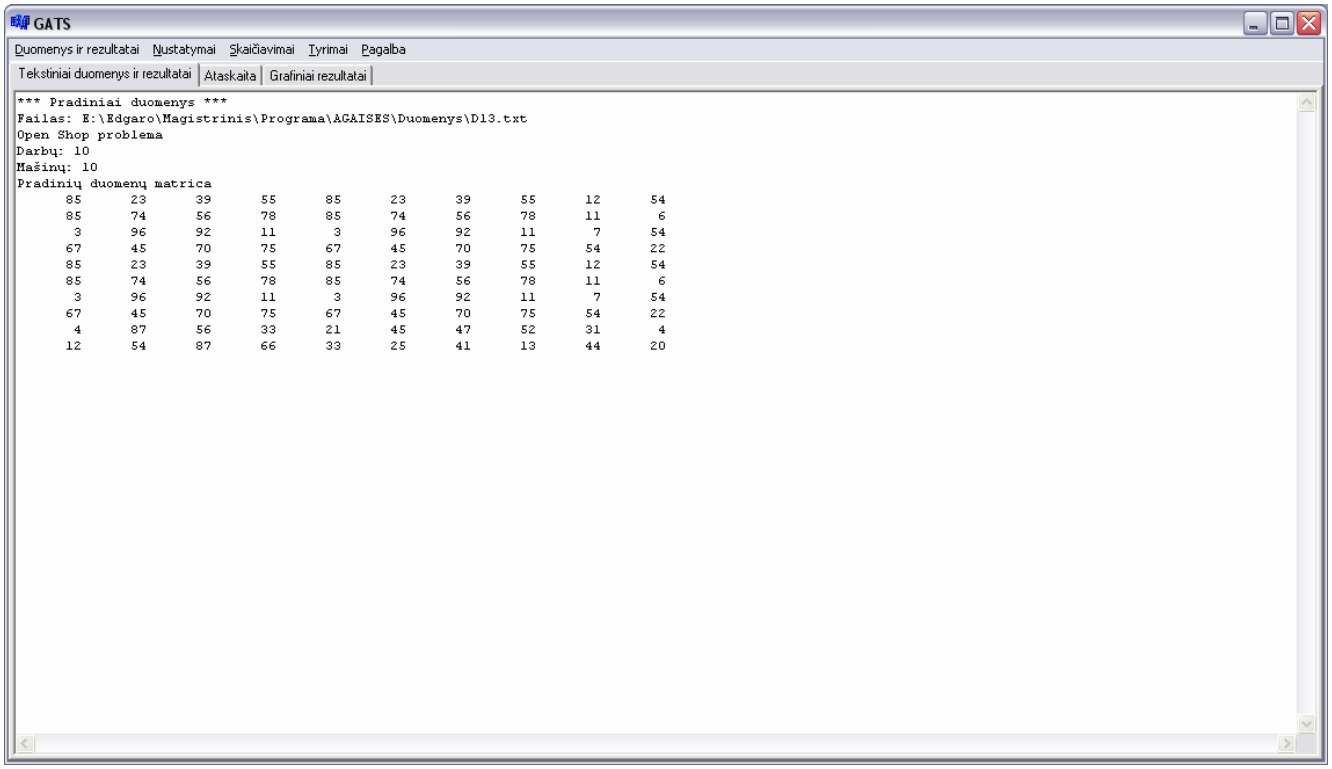
**Programos „GATS“ meniu**

| Meniu lygmuo | | | Paskirtis |
|---|---|---|---|
| **1 lygis** | **2 lygis** | **3 lygis** | |
| *Duomenys ir rezultatai* | *Įvesti duomenis* | *Iš failo* | Nuskaityti duomenis iš tekstinio duomenų failo. |
| | | *Generuoti atsitiktinius duomenis* | Generuoti atsitiktinius pasirinkto formato ir tipo duomenis. |
| | *Saugoti* | *Ataskaitą* | Išsaugoti programos ataskaitą tekstiniame faile. |
| | | *Tekstinius rezultatus* | Išsaugoti tekstinius rezultatus (optimalų tvarkaraštį) tekstiniame faile. |
| | | *Ganto diagramą* | Išsaugoti optimalaus tvarkaraščio Ganto diagramą. |
| | | *Algoritmo sprendinių įverčių grafiką* | Išsaugoti algoritmo sprendinių įverčių grafiką. |
| | *Išvalyti* | | Išvalyti programą nuo duomenų ir gautų rezultatų tam, kad būtų galima įvesti naujus duomenis. |
| | *Pabaigti programą* | | Pabaigti programos darbą. |
| *Nustatymai* | *Algoritmų parametrų nustatymai* | *Tikslios paieškos algoritmo* | Nustatyti tikslios paieškos algoritmo parametrus |
| | | *Atsitiktinio klaidžiojimo algoritmo* | Nustatyti atsitiktinio klaidžiojimo algoritmo parametrus |
| | | *Genetinio algoritmo* | Nustatyti genetinio algoritmo parametrus. |
| | | *Tabu paieškos algoritmo* | Nustatyti tabu paieškos algoritmo parametrus. |
| | *Tyrimų parametrų nustatymas* | *Genetinio algoritmo parametrų tyrimo* | Nustatyti genetinio algoritmo parametrų įtakos tyrimo parametrus. |
| | | *Algoritmų efektyvumo tyrimų* | Nustatyti algoritmų efektyvumo tyrimo parametrus. |
| *Skaičiavimai* | *Taikyti tikslų algoritmą* | | Paleisti pilnos paieškos algoritmą. |
| | *Taikyti atsitiktinio klaidžiojimo algoritmą* | | Paleisti atsitiktinės paieškos algoritmą. |
| | *Taikyti genetinį algoritmą* | | Paleisti genetinį algoritmą. |
| | *Taikyti tabu paieškos algoritmą* | | Paleisti tabu paieškos algoritmą. |
| *Tyrimai* | *Algoritmų efektyvumo tyrimas* | | Vykdyti pasirinktų algoritmų efektyvumo tyrimą. |
| | *Genetinio algoritmo parametrų įtakos tyrimas* | | Vykdyti genetinio algoritmo parametrų įtakos tyrimą. |
| *Pagalba* | *Programa* | | Programos pagalba. |
| | *Autorius* | | Informacija apie programos autorių. |

Įvedus tinkamo formato duomenų failą ar atsitiktinai sugeneravus (žr. 4.1 pav.) duomenis, programos „Tekstiniai duomenys ir rezultatai“ puslapyje sukuriama duomenų lentelė, kuri bus saugoma visos programos darbo metu (žr. 4.2 pav.).



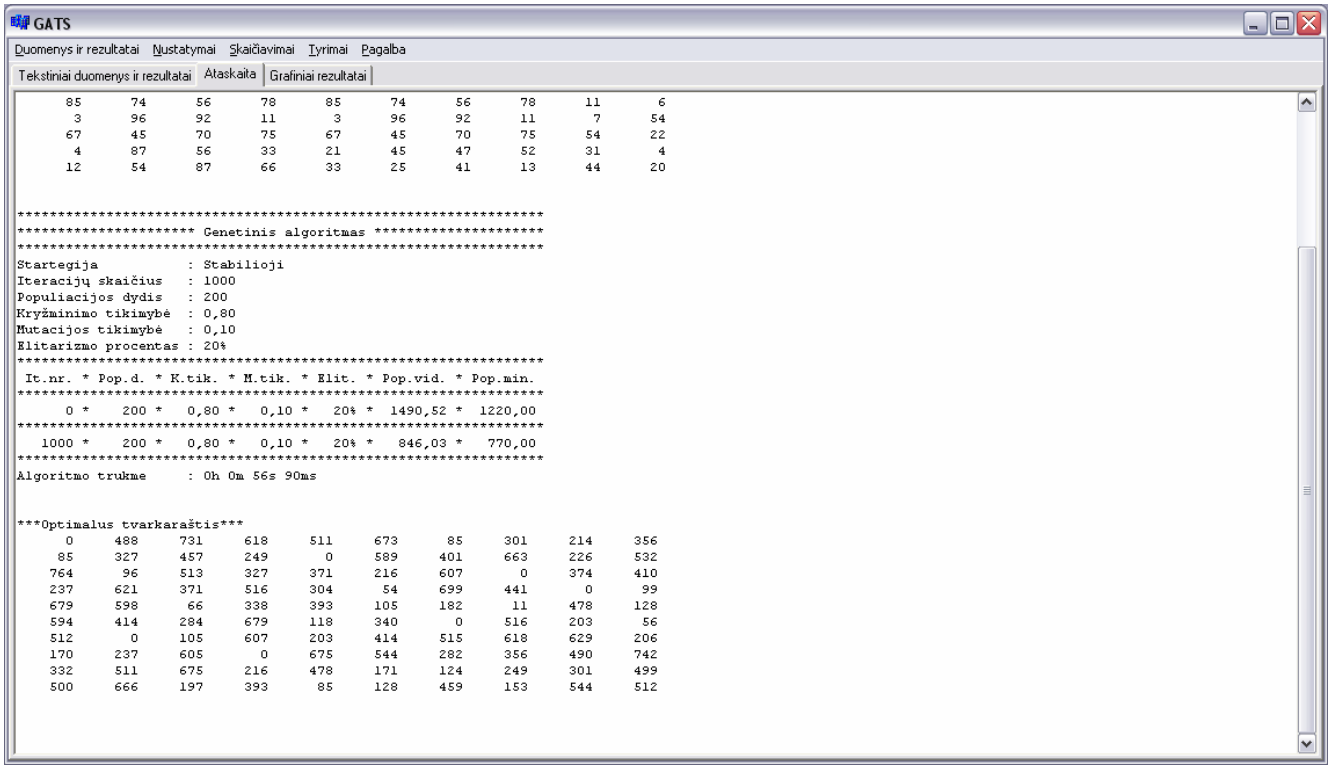

**4.1 pav. Programos „GATS“ atsitiktinių duomenų generavimo lango pavyzdys**




**4.2 pav. Programos „GATS“ duomenų lentelės lango pavyzdys**

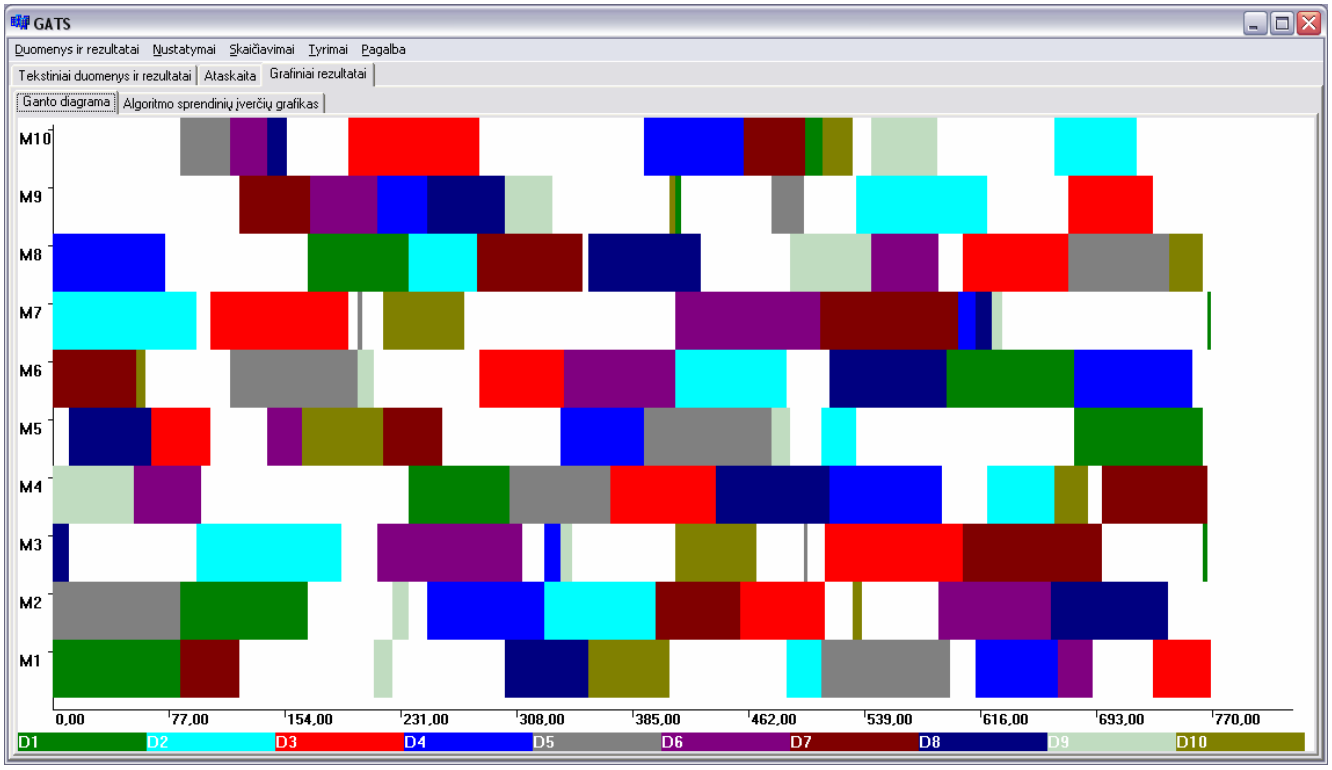
Atlikus norimus skaičiavimus (algoritmų taikymus ar jų taikymo tyrimus), programos lange „Ataskaita“ formuojama ataskaitą: į ją išvedami pradiniai duomenys, tarpiniai rezultatai bei gauti

rezultatai (optimalus tvarkaraštis (jis taip pat išvedamas „Tekstiniai duomenys ir rezultatai“ puslapyje) ar tyrimų suvestinės) (žr.4.3 pav.).

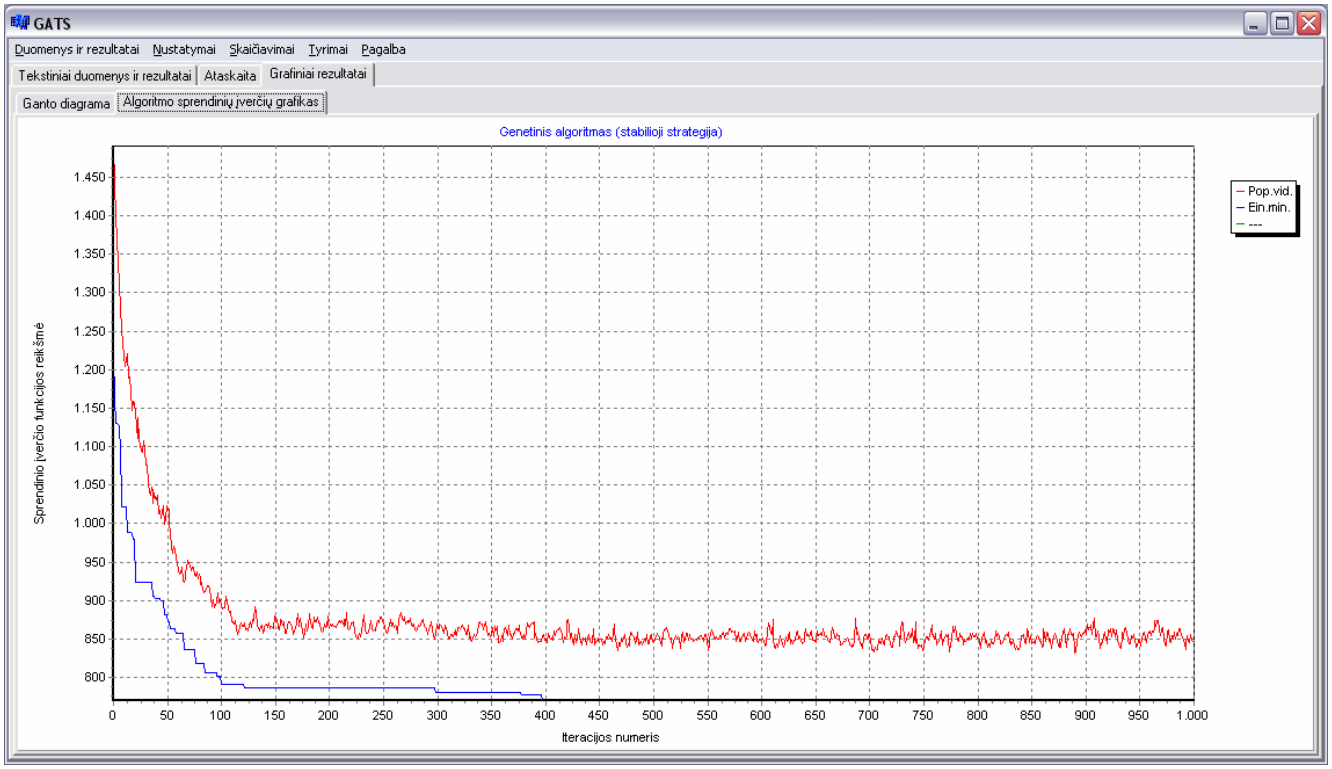



**4.3 pav. Programos „GATS“ ataskaitos išvedimo lango pavyzdys**

Programos lange „Grafiniai rezultatai“ pateikiama surasto optimalaus tvarkaraščio Ganto diagrama (žr. 4.4. pav.) ir taikyto algoritmo sprendinių įverčių grafikas (žr. 4.5 pav.).



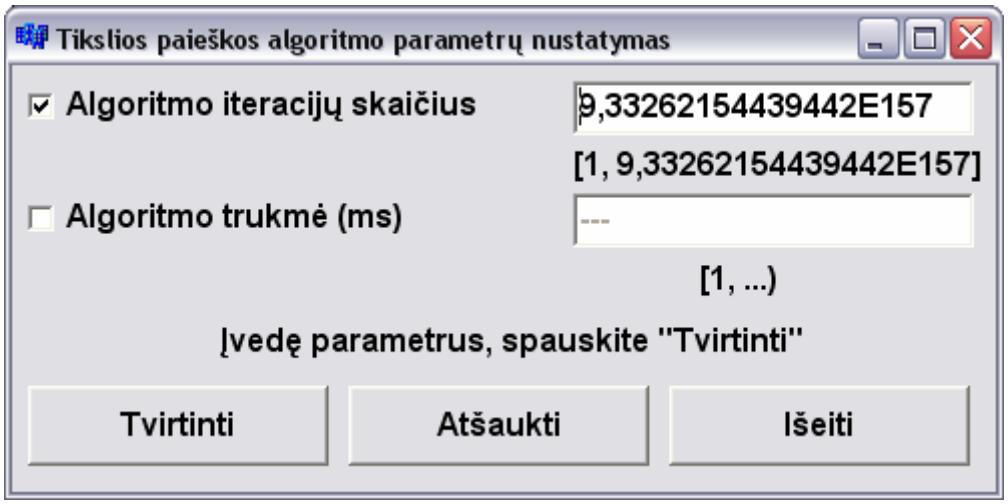

**4.4 pav. Programos „GATS“ grafinių rezultatų (Ganto diagramos) lango pavyzdys**



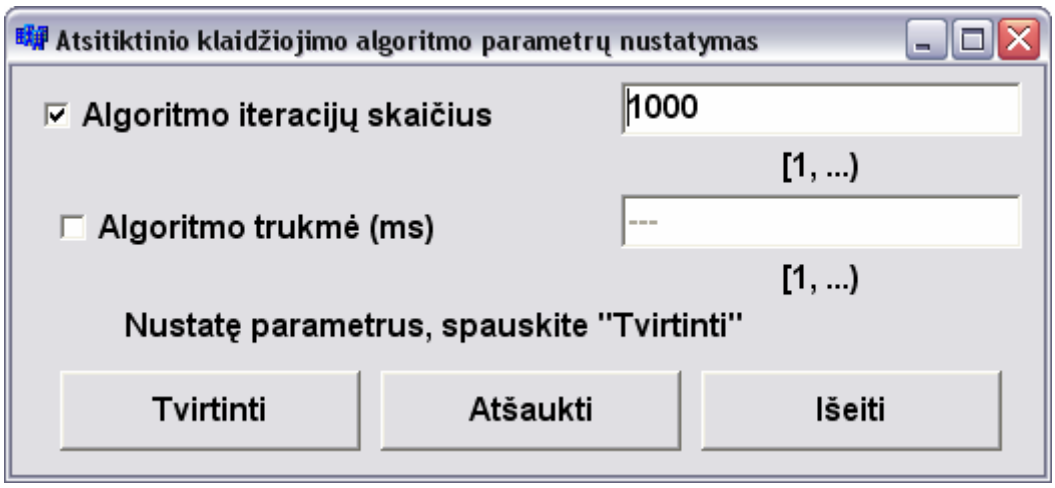

**4.5 pav. Programos „GATS“ grafinių rezultatų (sprendinių įverčių grafiko) lango pavyzdys**

Visus apskaičiuotus (tarpinius ir galutinius) rezultatus bei ataskaitą, vartotojas gali išsisaugoti tekstiniu (tekstiniams rezultatams) arba grafiniu (grafiniams rezultatams) formatu.

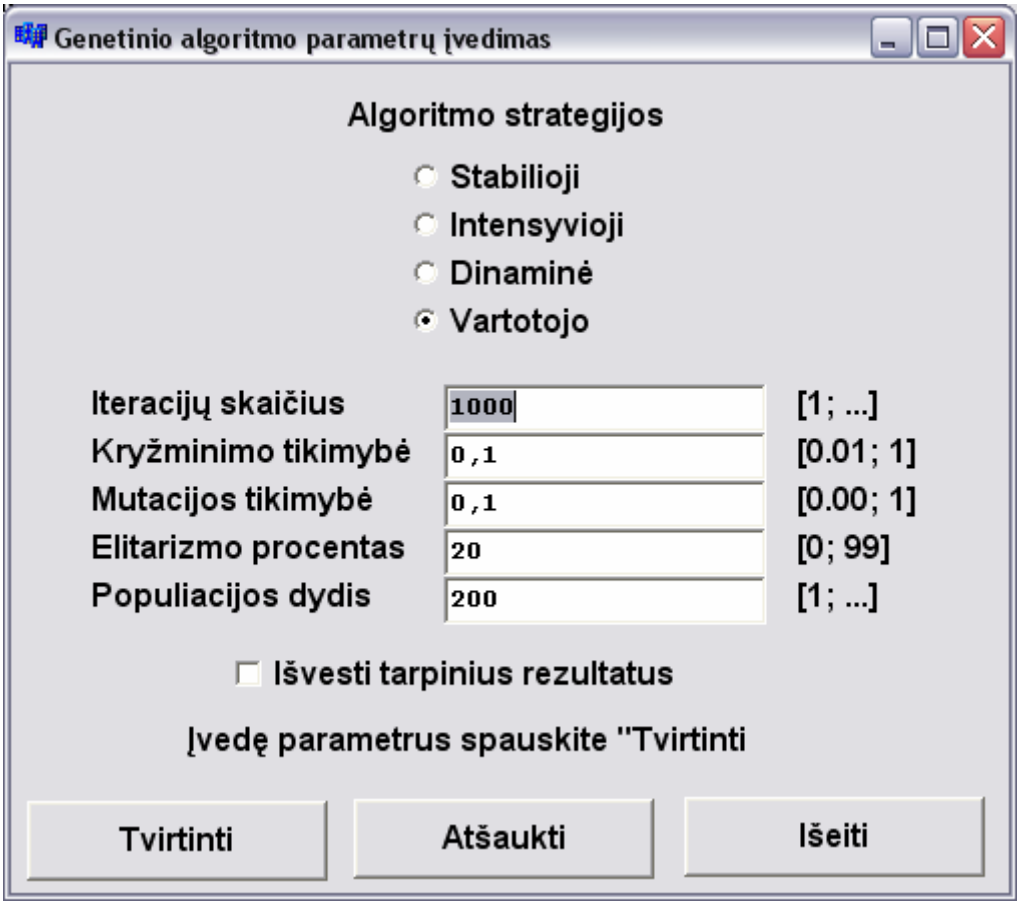
Prieš taikydamas norimą algoritmą, vartotojas gali pasirinkti jo parametrus: pilnosios paieškos algoritmo (žr. 4.6 pav.), atsitiktinės paieškos algoritmo (žr. 4.7 pav.), genetinio (žr. 4.8 pav.) ir tabu paieškos (žr. 4.9 pav.).



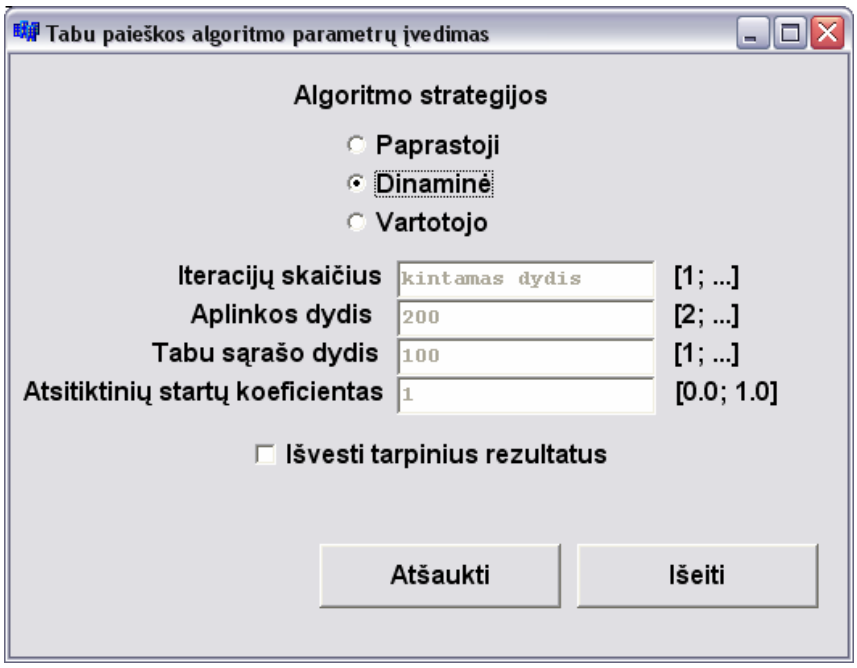

**4.6 pav. Programos „GATS“ pilnosios paieškos algoritmo parametrų nustatymo langas**




**4.7 pav. Programos „GATS“ atsitiktinės paieškos algoritmo parametrų nustatymo langas**



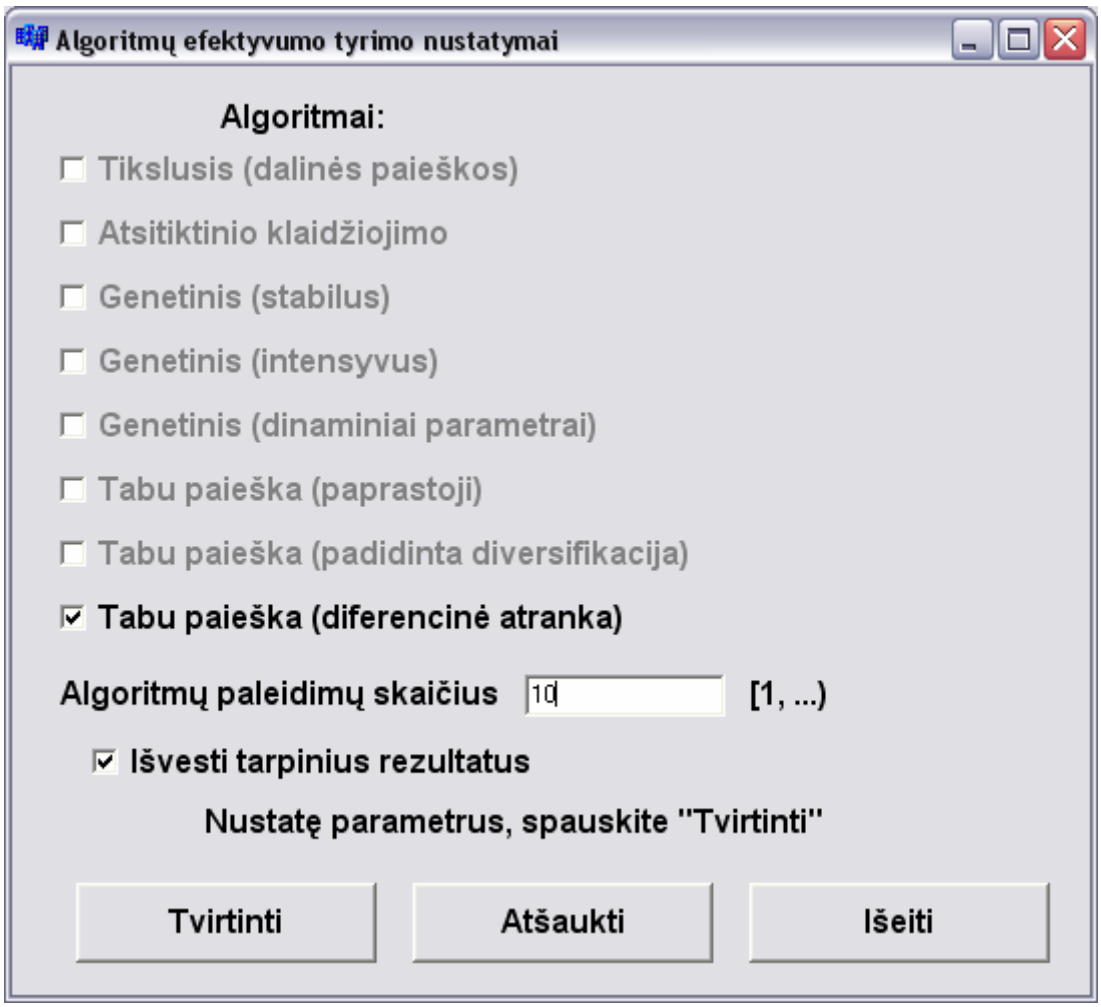

**4.8 pav. Programos „GATS“ genetinio algoritmo parametrų nustatymo langas**




**4.9 pav. Programos „GATS“ tabu paieškos algoritmo parametrų nustatymo langas**

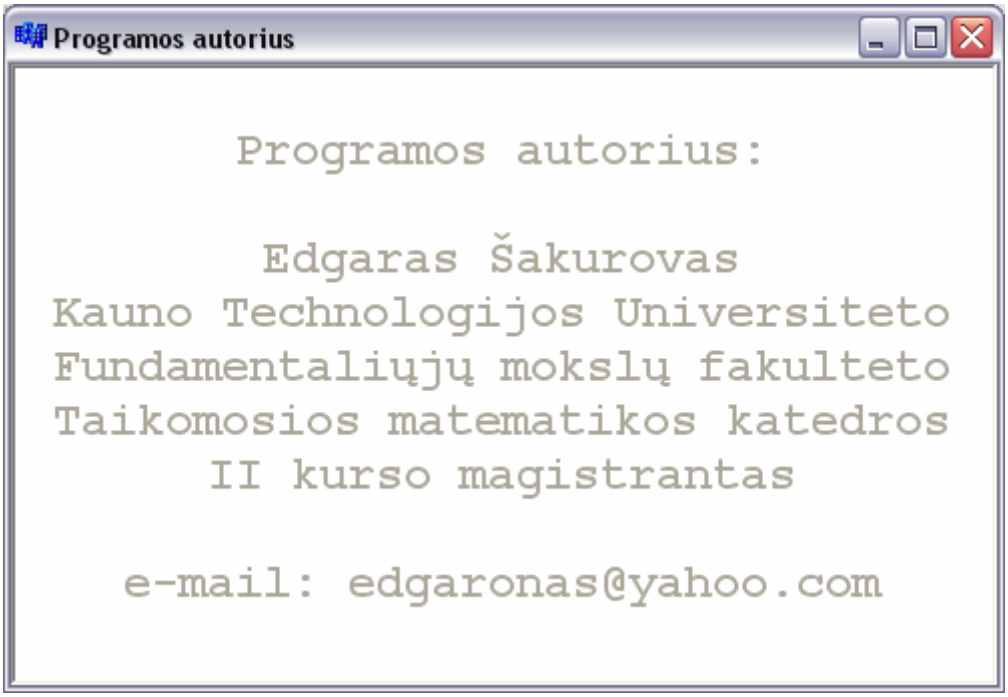
Norint atlikti tyrimus tiek genetinio algoritmo parametrų įtakos tyrimui (žr. 4.10 pav.), tiek algoritmo efektyvumo tyrimui (žr. 4.11 pav.), patartina iš anksto nusistatyti įvairias tyrimų parametrų reikšmes.


**4.10 pav. Programos „GATS“ genetinio algoritmo parametrų įtakos tyrimo nustatymo langas**

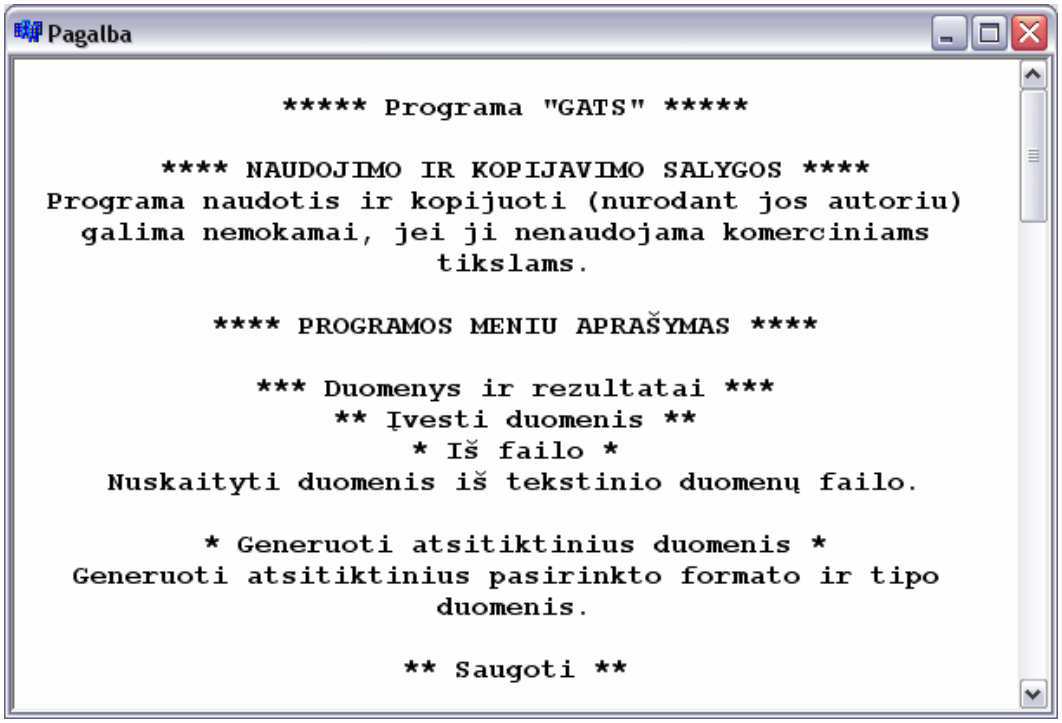



**4.11 pav. Programos „GATS“ algoritmų efektyvumo tyrimo parametrų nustatymo langas**

Meniu punkto „Pagalba“ klavišas „Autorius“ pateikia vartotojui duomenis apie programos autorių (žr. 4.12 pav.), o klavišas „Programa“ iškviečia programos pagalbos langą (žr. 4.13 pav.).




**4.12 pav. Programos „GATS“ autoriaus informacinis langas**

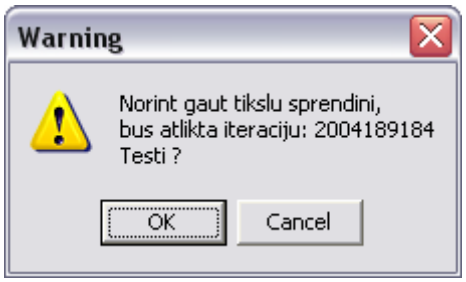



**4.13 pav. Programos „GATS“ pagalbos vartotojui langas**

## 4.2. REIKALAVIMAI PROGRAMAI IR JOS DUOMENIMS

Programa įgyvendinta „Borland C++ Builder 6“ kalboje. Nors programos failas „GATS.exe“, veikia ir aplinkoje, neturinčioje minėtos programinės įrangos. Reikalavimai operaciniai sistemai yra ne senesnė kaip „Windows XP“ versija, rezoliucijai – rekomenduojama, bet nebūtina, ne mažiau kaip 1024x768. mažesnės rezoliucijos ekranuose gali būti nekokybiškai pavaizduota Ganto diagrama.

Pilnosios paieškos sprendinio radimo atveju vartotojas įspėjamas apie visų galimų sprendinių skaičių (žr. 4.14 pav.), kurie bus skaičiuojami šio algoritmo metu, todėl didesnėms kaip 4×4 problemoms šio metodo taikyti nerekomenduojame.




**4.14 pav. Tikslaus sprendinio įspėjamojo pranešimo pavyzdys**

Duomenų failai turi atitikti tokius reikalavimus:

- Visų trijų problemų duomenų failuose pirmoje eilutėje turi būti įrašytas problemos pavadinimas anglų kalba, t.y. darbų fabrikui – Job Shop, srautiniam fabrikui – Flow Shop, atvirajam fabrikui – Open Shop. Jei nebus, bent vieno iš šių pavadinimo, ar jis bus įrašytas neteisingai, programa automatiškai užskaitys duomenų failą kaip netinkamą (nors jis gali būti ir geras).
- Antroje failo eilutėje turi būti įrašyti du sveikieji skaičiai, atskirti vienu bent tarpu ir atitinkantys: pirmasis – nagrinėjamos problemos mašinų skaičių, antrasis – darbų skaičių.
- Likusios eilutės (taip pat su neneigiamais sveikaisiais skaičiais) priklauso nuo nagrinėjamos problemos. Jeigu nagrinėjama problema yra atvirojo fabriko, tai turi būti parašytas sveikųjų arba realiųjų skaičių matrica, nurodytų antroje eilutėje matmenų. Negali būti tuščių reikšmių ir jokių simbolių, išskyrus tarpų, kurie skirti matricos elementų atskyrimui. Darbų fabriko problemoje, nelyginės matricos eilutės atitinka darbų laikų matricą, o lyginės – apribojimų matricą, atitinkamai pagal stulpelius. Vadinasi matricos eilučių turi būti dvigubai daugiau nei nurodyto antrojoje failo eilutėje. Srautinio fabriko problemoje visas formatas panašus kaip atvirojo fabriko problemos failų, tik po matricos turi būti įrašytas darbų eiliškumo stulpelis (žr. 4.2 lentelę).

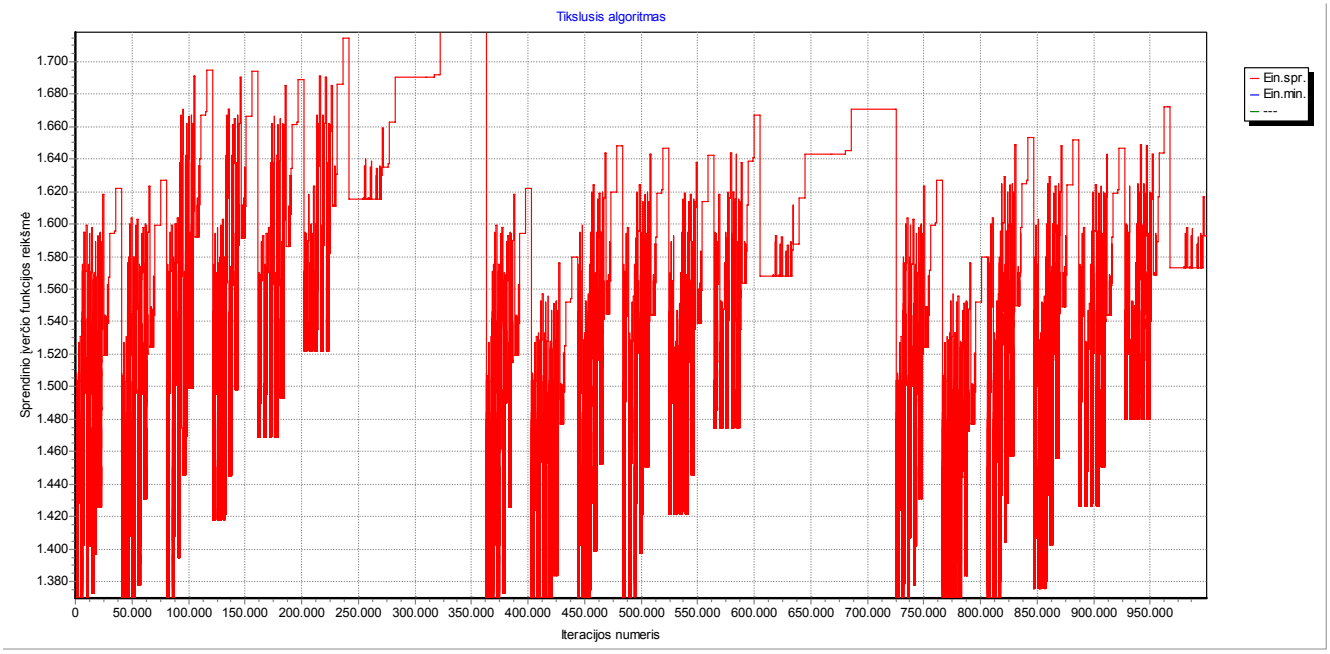
**4.2 lentelė**

**Programos tekstinių duomenų failų formatų pavyzdžiai**

| Atvirasis fabrikas | Darbų fabrikas | Srautinis fabrikas |
|---|---|---|
| Open Shop<br>m n<br>A11 A12 ... A1n<br>A21 A22 ... A2n<br>... ... ... ...<br>Am1 Am2 ... Amn | Job Shop<br>m n<br>A11 A12 ... A1n<br>C11 C12 ... C1n<br>A21 A22 ... A2n<br>C21 C22 ... C2n<br>... ... ... ...<br>Am1 Am2 ... Amn<br>Cm1 Cm2 ... Cmn | Flow Shop<br>m n<br>A11 A12 ... A1n<br>A21 A22 ... A2n<br>... ... ... ...<br>Am1 Am2 ... Amn<br>Nr1<br>Nr1<br>Nr3<br>...<br>Nrm |

# 5. DISKUSIJA

Tvarkaraščių sudarymo problema priklauso NP uždavinių sunkumo klasei. Tai akivaizdžiai patvirtina 3.4 skyrelyje minėtas tikslaus algoritmo panaudojimas kelioms atvirojo fabriko problemoms. Nesunkiai iš 3.2 lentelės ir 3.1 grafiko tendencijų pastebime, jog problemai didėjant algoritmo trukmė neapibrėžtai auga. Taigi netgi palyginus nesudėtingoms problemoms (4 × 4 formato) rasti tikslų sprendimą įmanoma (teorinis laikas pateikiamas), bet laiko sąnaudos yra akivaizdžiai per didelės. Palyginimui genetinis algoritmas su stabiliaja strategija vidutiniškai trunka apie 12 sekundžių, o tabu paieškos apie 3,5 sekundės. Ir abu daugumai atvejų (bent jau tokio dydžio problemoms) randa artimų minimumui sprendinių per gerokai priimtinesnį laiką. Atlikti dalinę paiešką taip pat netikslinga, nes ji determinuota ir struktūrinė, vadinasi, priklauso tik nuo iteracijų skaičiaus (žr. 5.1 pav.). Tai tik akivaizdžiai patvirtina metaeuristikų populiarumo ir aktyvaus vystymosi priežastis.




**5.1 pav. Dalinės (tikslios) paieškos algoritmo sprendinių įverčių grafiko pavyzdys 10 x 10 formato Open Shop problemai (1000000 iteracijų)**

Atsitiktinis klaidžiojimas, kaip matyti iš 3.2 paveikslo, taip pat mažai naudingas netgi dideliam iteracijų skaičiui (pavyzdžiui, artimam pilnos paieškos iteracijų skaičiui), kadangi yra tikimybė aplankyti tą pati sprendinį kelis kartus.

## 5.1. GENETINIO ALGORITMO PARAMETRAI

2 skyriaus 2.2 lentelėje pateiktus genetinio algoritmo parametrų siūlymus galime pakomentuoti taip:

- De Jong – didesnė paieškos sritis; dirbtinė elitarizmo strategija; stabilus palyginus su kitais dviem algoritmais.

- Schaffer – mažesnė paieškos sritis, bet didesnis paieškos spektras, tikėtinas nestabilus populiacijos pajėgumas.
- Grefenstette – intensyvus algoritmas, pasižymintis antro siūlymo savybėmis.

3.5.1 skyrelyje minėto tyrimo rezultatai priveda prie tokių samprotavimų: pastebime, jog kintant kryžminimo tikimybei bendrasis populiacijos pajėgumas išlieka stabilus, tačiau minimumai akivaizdžiai gerėja (3.2 pav. a dalis). Visgi, jeigu nenaudojama elitarizmo strategija, kryžminimo tikimybė turėtų būti $K < 1$. Didinant mutacijos tikimybę (b dalis), tiek populiacijos pajėgumas, tiek surastas minimumas gerėja, tačiau taip yra dėl elitarizmo strategijos naudojimo. Esant pradiniam parametrų parinkimui geriausia, kai visos chromosomos „dalyvauja“ populiacijos reprodukcijoje (e dalis). Tačiau gali būti situacijų, kai geriausi sprendiniai nepaklius į sekančią kartą, todėl $E > 0\%$ (t.y. bent vienas geriausias sprendinys išsaugomas). Kuo didesnis iteracijų skaičius (arba populiacijos dydis), tuo daugiau šansų, kad algoritmo metu bus išnagrinėta didesnė sprendinių erdvės dalis. Rasti minimumai didėjant iteracijų (populiacijos dydžiui) skaičiui, turi tendenciją didėti (c ir d dalys). Tačiau šie parametrai tiesiogiai priklauso ir nuo laiko sąnaudų, kas realiose situacijose verčia juos palikti fiksuotus arba priklausomus nuo užduoties sunkumo.

## 5.2. GENETINIO IR TABU PAIEŠKOS ALGORITMŲ STRATEGIJOS

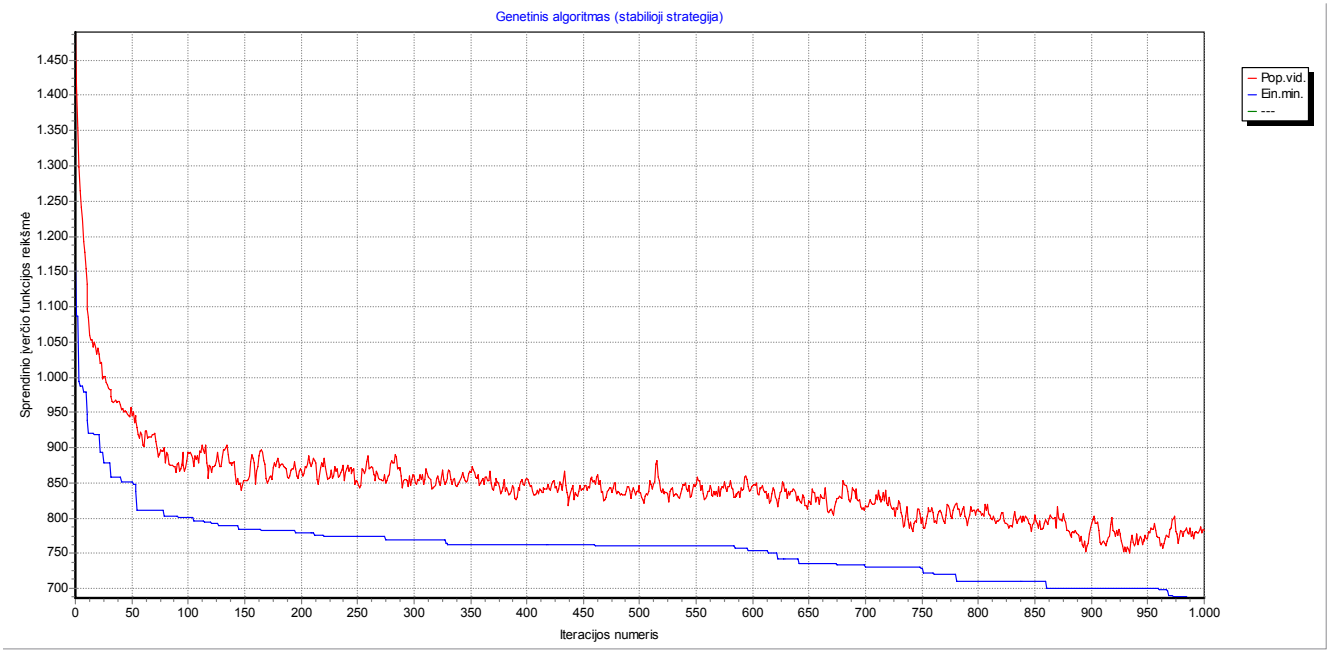
Atlikus praeitame skyrelyje minėtus tyrimus bei padarius tam tikrus patobulinimus nuo paskutinių rezultatų (plačiau žr. [18], [35]), jų rezultatus galime reziumuoti tokia 5.1 lentele, kurioje siūlomos genetinio algoritmo parametrų parinkimo strategijos.

**5.1 lentelė**

**Siūlomos genetinio algoritmo strategijos (algoritmo parametrų atžvilgiu)**

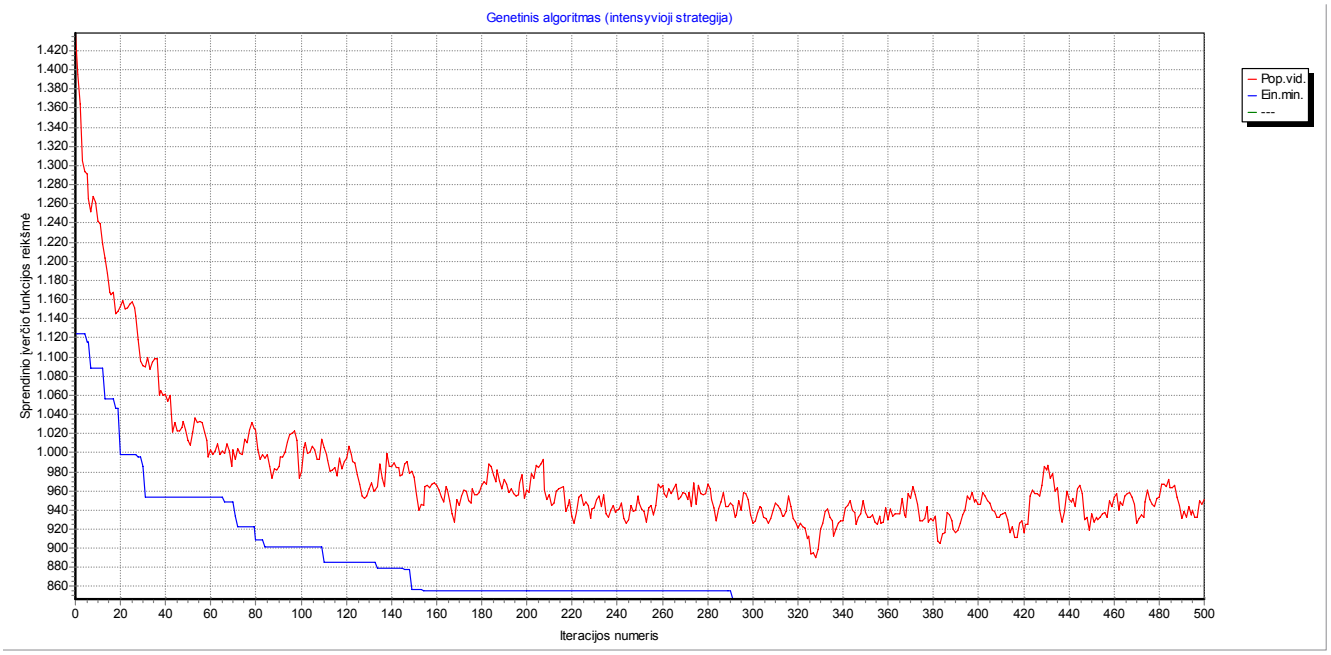
| Nr. | Strategija | *IterSk* | *P* | *K* | *M* | *E*, % |
|---|---|---|---|---|---|---|
| 1 | Stabilioji | priklausomas nuo problemos dydžio (pvz., $10 \cdot m \cdot n$) | $2 \cdot m \cdot n$ | ~0.8 | ~0.1 | ~20 |
| 2 | Intensyvioji | priklausomas nuo problemos dydžio (pvz., $5 \cdot m \cdot n$) | $m \cdot n$ | ~0.9 | ~0.3 | ~5 |
| 3 | Dinaminė | skaičius nustatomas algoritmo eigoje (pagrindas $10 \cdot m \cdot n$) | $2 \cdot m \cdot n$ | tikimybė nustatoma algoritmo eigoje (pagrindas 0.8) | tikimybė nustatoma algoritmo eigoje (pagrindas 0.1) | ~50% (tam, kad dirbtinai nepablogintų daugumos narių) |

Stabilios strategijos atveju išlieka stabilūs populiacijos pajėgumas ir minimumas (žr. 5.1 pav.). Laiko sąnaudos priklauso nuo problemos sudėtingumo ir iteracijų skaičiaus parinkimo. Garantuojamas geras sprendinys per baigtinį laiką.




**5.2 pav. Genetinio algoritmo stabilios strategijos sprendinių įverčių grafiko pavyzdys 10 x 10 formato Open Shop problemai**

Intensyvios strategijos atveju populiacijos pajėgumas nėra stabilus. Laiko sąnaudos gerokai mažesnės nei stabilios strategijos, tačiau sprendinio kokybė sunkiai prognozuojama (algoritmas gali pateikti tiek tikslesnius, tiek mažiau tikslius atsakymus).




**5.3 pav. Genetinio algoritmo intensyvios strategijos sprendinių įverčių grafiko pavyzdys 10 x 10 formato Open Shop problemai**

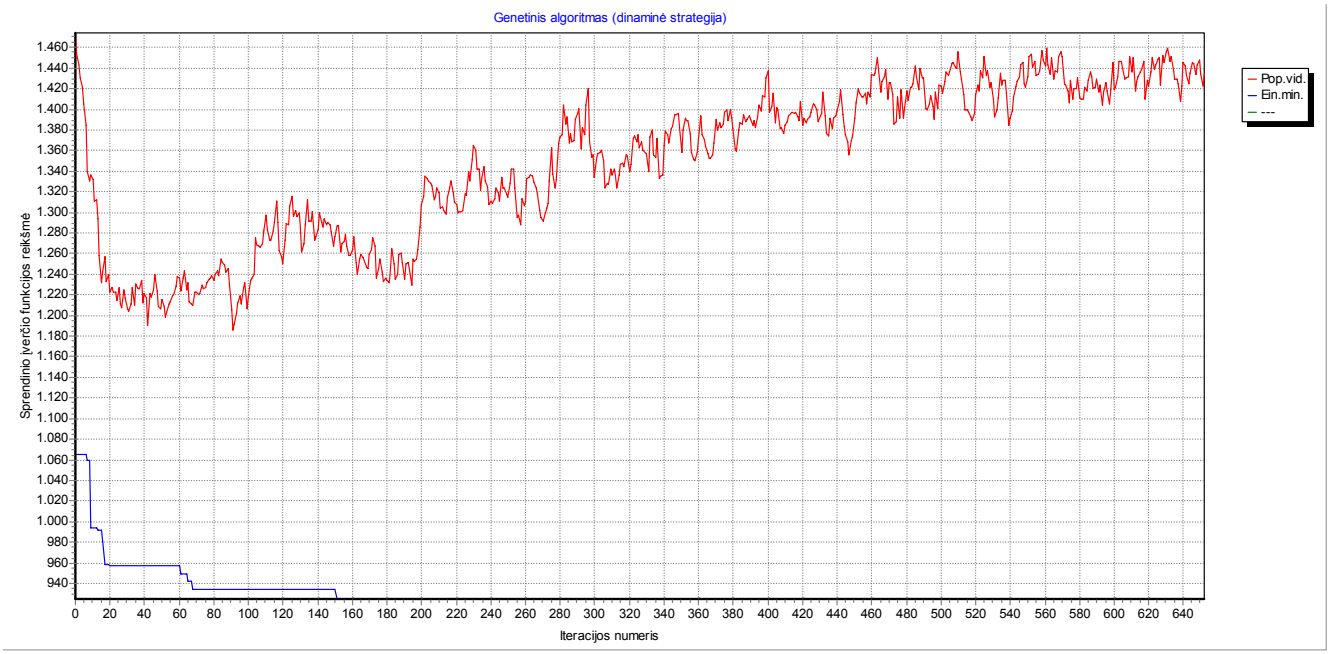
Būtent po intensyviosios genetinio algoritmo strategijos taikymo 5.1 lentelės 3 eilutės paskutiniame stulpelyje įrašyta ~50%, t. y. didelė kryžminimo ir mutacijos tikimybė gali sugadinti daugumos populiacijos narių pajėgumą, tokiu būdu prarandamas gerų savybių paveldėjimo efektas ir algoritmas paprasčiausiai diverguoja (žr. 5.4 pav.). Tačiau padidinus mutacijos lygį, t.y. sukeičiami ne du, o daugiau elementų, algoritmo divergavimas kiek pristabdomas, ir tuo pačiu padidinama diversifikacija, vadinasi, didesnė tikimybė aptikti naujus dar neaplankytus sprendinius. Tačiau vis dėlto paveldėjimo efekto dar nėra. Jam užtikrinti tik pusė populiacijos narių su blogesne pajėgumo funkcijos reikšme keičiami. Paskutinis patobulinimas paremtas atsitiktinių startų privalumu, naudojamų paprastoje Tabu paieškoje. Kurių metu vėlgi apie 50 procentų prastesnių populiacijos narių sugeneruojama iš naujo. Vadinasi, galime reziumuoti šią strategiją ir jos parametrus:

$$K = 0.8 + \frac{kitMin}{iterSk} \cdot 0.15 + \frac{einIt}{iterSk} \cdot 0.05,$$

$$M = 0.05 + \frac{kitMin}{iterSk} \cdot 0.8 + \frac{einIt}{iterSk} \cdot 0.05,$$

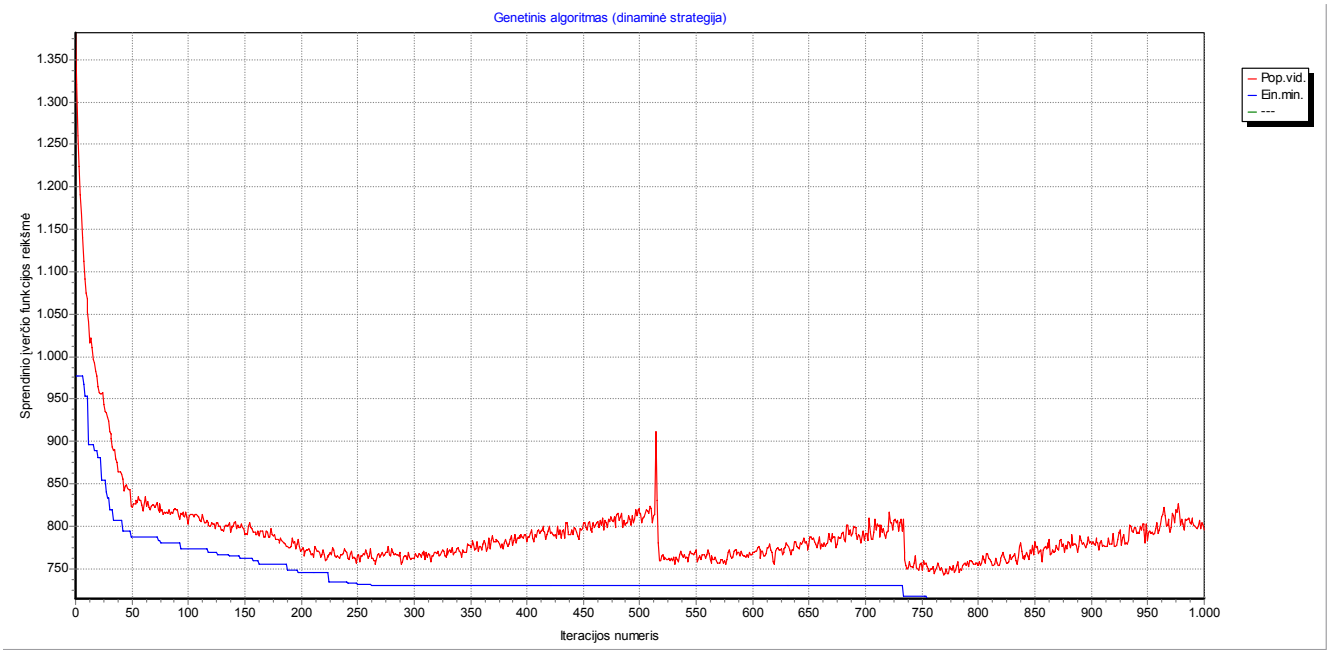
$$Mlygis = 1 + f_{floor}\left(\frac{kitMin \cdot c}{2 \cdot iterSk} - 1\right),$$

čia *kitMin* – geriausio surasto sprendinio stabilios būsenos iteracijose skaičius, *einIt* – einamoji algoritmo iteracija, *iterSk* – maksimalus iteracijų skaičius (jei algoritmas nebus nutrauktas), $f_{floor}$ - funkcija, kuri grąžina tik sveikąją dalį. Jau minėtas specialus kintamasis *Mlygis* yra mutacijos lygis, t.y. kiek kartų sprendiniui bus pritaikyta dviejų elementų sukeitimo mutacija su tikimybe *M*. Algoritmas sugeneruoja naujus sprendinius, jei *kitMin* daugiau nei ketvirtadalis visų iteracijų.



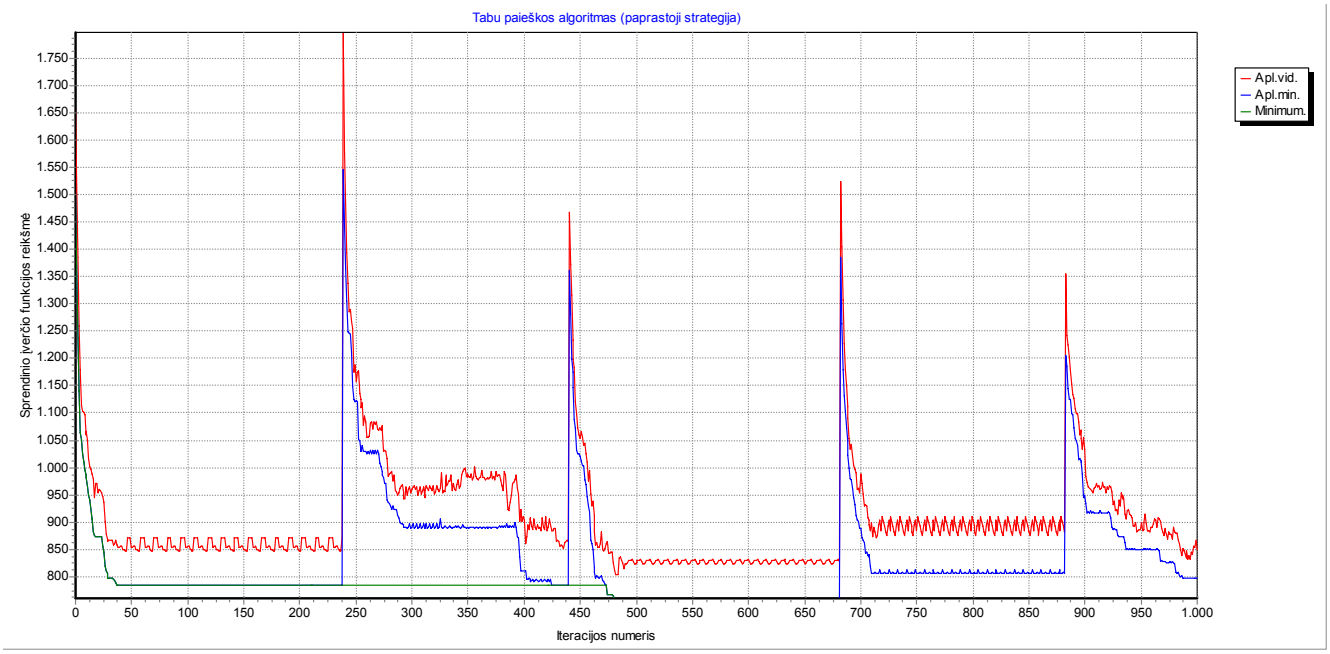

**5.4 pav. Genetinio algoritmo dinaminės strategijos sprendinių įverčių grafiko pavyzdys 10 x 10 formato Open Shop problemai be elitarizmo ir atsitiktinių starto modifikacijų**

Trumpai strategijos esmė gali būti apibūdinta taip: iš pradžių algoritmas yra su kiek padidinta intensifikacija, iteracijoms didėjant intensifikacija mažėja, didėja diversifikacija. Jei geriausias sprendinys nepagerinamas, intensifikacija sumažėja iki minimumo, o diversifikacija padidėja iki maksimumo. Kaskart (kas kelias ar keliasdešimt iteracijų) sprendiniui gerėjant, intensifikacija atitinkamai didinama, diversifikacija mažinama. Jei algoritmą ištiko stagnacija – sugeneruojami atsitiktiniai sprendiniai (žr. 5.5 pav.)



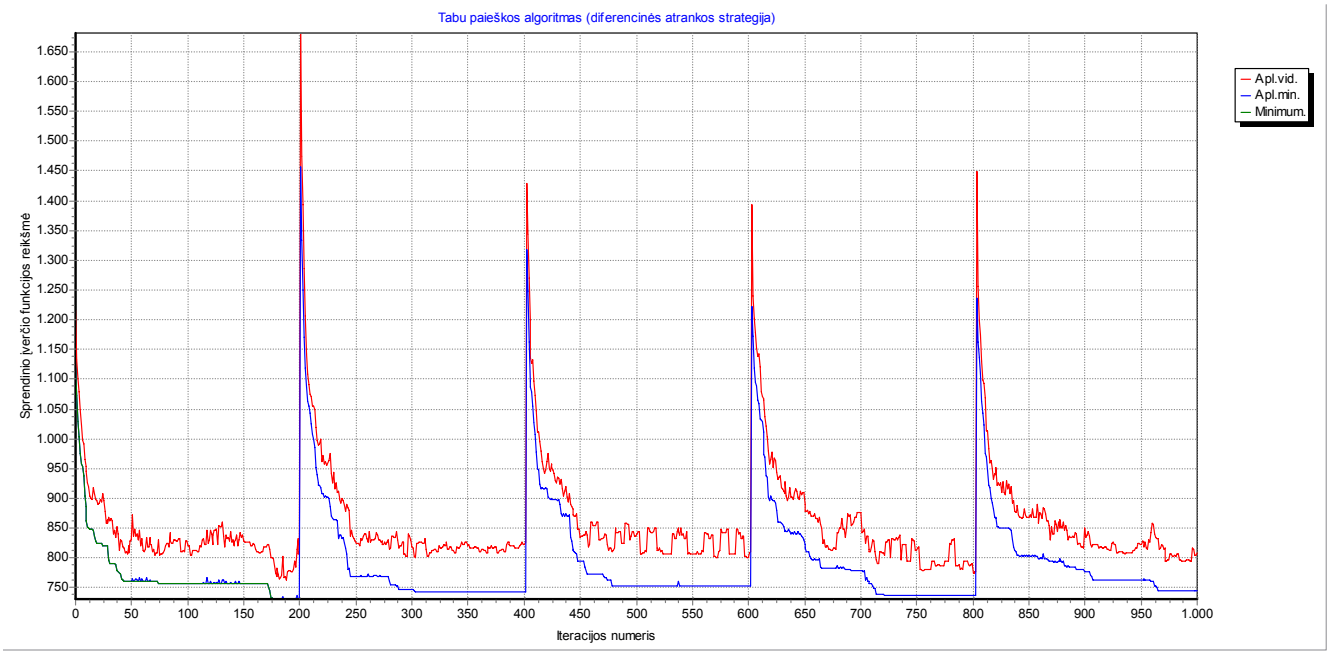

**5.5 pav. Genetinio algoritmo dinaminės strategijos sprendinių įverčių grafiko pavyzdys 10 x 10 formato Open Shop problemai su elitarizmo ir atsitiktinių starto modifikacijomis**

Kalbant apie tabu paieškos strategijas, kaip minėta teorinėje dalyje, pats tabu įgyvendinimas nėra vienareikšmis, todėl pasirinkta elementari, taip vadinama „paprastoji“ strategija, kurioje aplinkos dydis priklauso nuo problemos, iteracijų skaičius taip pat, tabu sąrašo ilgis „magiška“ 7 reikšmė. Algoritmo metu padaromi keli atsitiktiniai startai (žr. 5.6 pav.). Tačiau jau pastarajame paveiksle matyti, kad pavaizduotai problemai tabu sąrašas yra per mažas, ir algoritmas turi ciklų. Vienas iš elementarių sprendimo būdu yra tabu sąrašo padidinimas (pavyzdžiui, pusę aplinkos dydžio), tačiau prastesni algoritmo rezultatai yra dar dėl to, jog naudojama sprendinio aplinka, kurioje sprendiniai mažai skiriasi vienas nuo kito (tik dvejomis kodavimo pozicijomis), todėl algoritmo konvergavimas lėtas. Tai būtų galima spręsti aplinkos bei tabu sąrašo padidinimu, tačiau iškyla neefektyvaus resursų naudojimo klausimas.




**5.6 pav. Paprastosios tabu paieškos algoritmo sprendinių įverčių grafiko pavyzdys 10 x 10 formato Open Shop problemai**

Siekiant išvengti artimos (Hemingo atstumo prasme) aplinkos ir limituoto tabu sąrašo ilgio trūkumų, buvo pasirinktas kiek modifikuotas geriausio sprendinio-kandidato atrankos metodas. Jo esmė yra ta, kad kandidato įverčio funkcijai suteikiama tam tikra bauda, kuo daugiau kandidatas skiriasi nuo globalaus minimumo, tuo mažesnis jo įvertis ir atvirkščiai. Vadinasi, tokiu būdu į atranką paklius ne tik geri, bet lokaliai mažai besiskirianti sprendiniai, tačiau ir blogesni, kurie tolimesni – taip padidinant diversifikaciją (žr. 5.7 pav.)

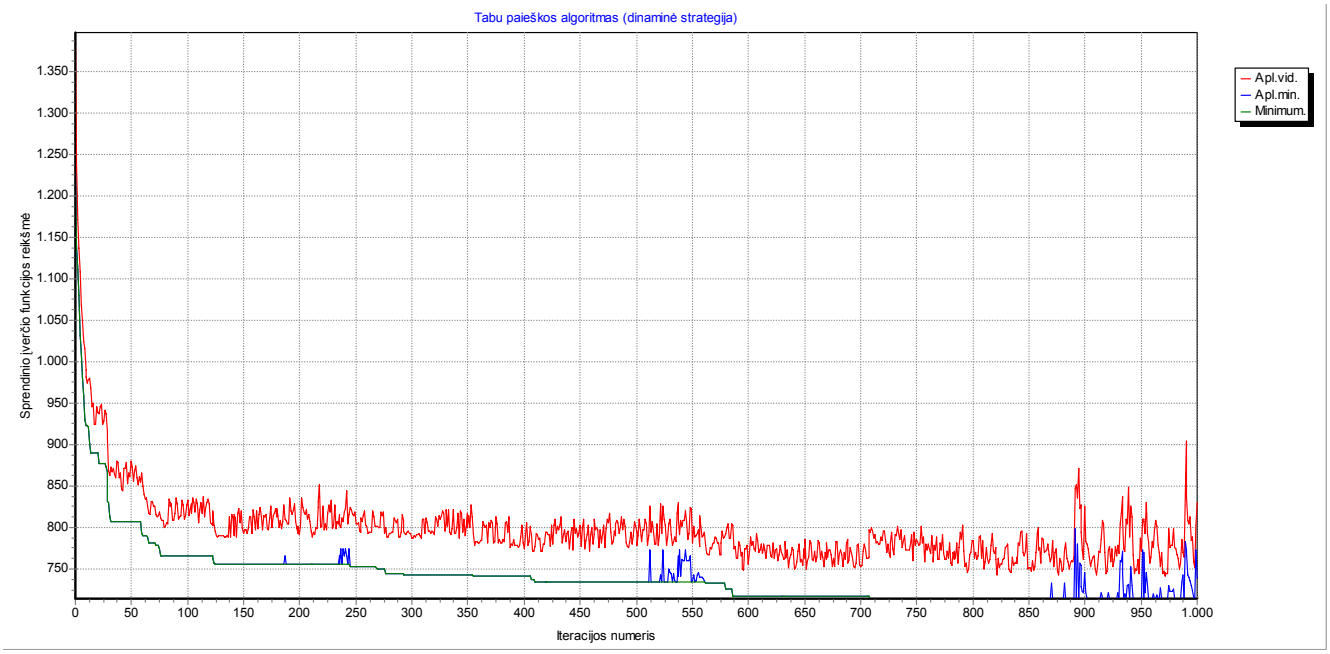



**5.7 pav. Tabu paieškos su diferencine atranka algoritmo sprendinių įverčių grafiko pavyzdys 10 x 10 formato Open Shop problemai**

Prieš tai minėtų tabu paieškos algoritmo trūkumų sprendimui pasirinkome aplinkos sudarymo modifikaciją. Visiems sprendinio aplinkos nariams taikomas dviejų ir daugiau elementų sukeitimas. Tokiu būdu visi elementai nutolsta Hemingo atstumo prasme >2, t.y. atliekamas dirbtinis aplinkos postūmis. Jei sukeičiame 3 aplinkos narių elementus, Hemingo postūmis bus >3 – postūmis didėja – diversifikacija taip pat. Norime pastebėti, jog akivaizdu, kad ne visi elementai bus >2, t.y. atsiras identiškų einamajam sprendiniui jo aplinkos narių, kadangi atliekamas vienodas elementų keitimas visai aplinkai. Tačiau šią situaciją lengvai išsprendžia tabu sąrašas. Tokiu būdu,

$$Klygis = f_{ceil}\left(\frac{kitMin \cdot c}{iterSk}\right),$$

kur *Klygis* aplinkos narių elementų sukeitimų skaičius, $f_{ceil}$ – apvalinimo funkcija, kuri grąžina sveikąją dalį +1. Tabu paieškos su padidinta diversifikacija pavyzdys pateiktas 5.8 pav.




**5.8 pav. Tabu paieškos su padidinta diversifikacija algoritmo sprendinių įverčių grafiko pavyzdys 10 x 10 formato Open Shop problemai**

3.5.2 skyrelyje minėtą algoritmų efektyvumo tyrimą, naudojant penkias strategijas: genetinio algoritmo stabiliąją, genetinio algoritmo intensyviąją, genetinio algoritmo dinaminę ir tabu paieškos algoritmo paprastąją bei padidintos diversifikacijos; taip pat du papildomus algoritmus – tikslios ir atsitiktinės paieškos – reziumuojame taip:

- Populiacijos (ar sprendinio aplinkos) atžvilgiu, visi tabu algoritmai pradeda paiešką su geriausiomis aplinkomis (tabu paieškos su diferencine atranka aplinka vidutiniškai šiek tiek geresnė), visos genetinio algoritmo strategijos pradeda su panašia, tačiau prastesne populiacija, kurios stabilumas yra geresnis. Tą lemia tai, jog genetiniai algoritmai generuoja

tam tikrą skaičių atsitiktinių skirtingų sprendinių, kai tuo tarpu tabu paieškoje sugeneruojamas vienas sprendinys, jam ir sudaroma aplinka.

- Pradinės aplinkos minimumų atžvilgiu geresni pradiniai sprendiniai genetinio algoritmo strategijose dėl tų pačių priežasčių, minėtų praeitame punkte. Palyginimui (3.4 pav. b) dalis) tikslios paieškos pradinis iš anksto determinuotas sprendinys yra geresnis nei atsitiktinės paieškos pradinis atsitiktinis sprendinys.
- Geriausias galutinės populiacijos sprendinių vidurkis yra dinaminėje genetinio algoritmo strategijoje (tai daugiausiai lemia 50% elitarizmo strategija), visi likusieji yra kiek prastesni (išskyrus paprastąja tabu paiešką, kurios galutinė aplinka pati prasčiausia).
- Gautų minimumų atžvilgiu vėlgi nugalėtoja skelbiame tabu paiešką su padidinta diversifikacija, antra – kaip ir tikėtasi, dinaminė genetinio algoritmo strategija, trečia tabu paieška su diferencine atranka, ketvirta – intensyvioji genetinio algoritmo strategija, toliau paprastosios genetinio ir tabu paieškos algoritmų versijos (nors pirmoji gerokai lenkia pastarąją, tačiau tik šiek tiek lenkia dalinę paiešką, kurios trukmė apytiksliai pusantro karto ilgesnė), patys prasčiausi atsitiktinės paieškos surasti minimumai.
- Tačiau, jei kalbame apie algoritmų trukmės sąnaudas, pirmaujančios yra genetinio algoritmo intensyvioji, paprastoji tabu ir tabu paieška su padidinta diversifikacija. Daugiausiai laiko sąnaudų reikalauja genetinio algoritmo strategija.
- Paskutiniame punkte – pradinio ir gauto minimumo skirtumų – akivaizdžiai pirmauja tabu paieška su padidinta diversifikacija, po jos tabu paieška su diferencine atranka. Atsitiktinės paieškos pokytis geras, tačiau jos paieška pradedama ne nuo lokalaus (ar bent jau galimo mažiausio) sprendinio. Kaip ir tikėtasi, mažiausias pokytis yra stabiliojoje genetinio algoritmo strategijoje.

# 6. IŠVADOS

Genetinio algoritmo parametrai:

- chromosomų kryžminimas ir mutacija yra esminiai genetinio algoritmo operatoriai sprendinių sklaidos ir algoritmo konvergavimo atžvilgiu. Įvairios jų tarpusavio kombinacijos gali gerokai pagreitinti algoritmo konvergavimą, tačiau tam taip pat reikalingi papildomi komponentai;
- elitarizmo procentas iš esmės nėra toks kritinis ir jis veikia daugiau kaip „buferinis“ parametras norint būsimose populiacijoje neprarasti geriausio iš surastų lokalių minimumų, tačiau agresyvios algoritmo diversifikacijos atveju šis parametras lemia vieną iš svarbiausių genetinio algoritmo idėjų – gerų savybių paveldėjimo efektą;
- iteracijų skaičius ir populiacijos dydis mažiau svarbūs teoriškai, bet labai svarbūs praktiškai. Jie tarpusavyje susiję ir turi būti siejami su sprendžiamo uždavinio kodavimu arba skaičiavimo resursais;
- gamybinių tvarkaraščių sudarymo atveju, kuomet naudojama įverčio funkcija, įtraukianti visus apribojimus; optimalių parametrų parinkimas galimas, tačiau tinkamas dinaminis (priklausomas nuo problemos ir/arba algoritmo metu gaunamų sprendinių) parametrų kitimas yra efektyvesnis resursų ir/arba surasto optimalaus sprendinio kokybės atžvilgiu.

Tabu paieška:

- per didelė laisvė algoritmo įgyvendinimui turi privalumų, tačiau kartu ir trūkumų, susijusių su tinkamo metodo atitinkamai problemai pasirinkimo teisingumu;
- palyginus su kanoniniu genetiniu algoritmu paprastoji tabu paieška pateikia mažesnės kokybės sprendinius, tačiau tabu paieška su tam tikromis modifikacijomis (šiuo atveju su padidinta diversifikacija ir diferencine) yra efektyvūs ir daug žadantis metodas.

Genetiniai algoritmai ir tabu paieška:

- tinkamai parinkus parametrus bei algoritmų įgyvendinimo metodus abu tiek genetinis, tiek ir tabu paieškos algoritmai yra tinkami tvarkaraščių sudarymo problemai, tačiau tikėtina, jog visgi geriausias ne kuris nors iš šių algoritmų, o tam tikras jų abiejų junginys, t.y. vienas algoritmas savo geromis savybėmis tam tikrose situacijose kompensuoja kito algoritmo prastąsias savybes tose situacijose ir atvirkščiai.

# 7. REKOMENDACIJOS

Genetinio algoritmo modifikacija – dinaminė strategija – paremta tabu paieškos algoritme naudota atsitiktinių startų idėja. Tačiau, jeigu genetinio algoritmo visai einamajai sprendinių populiacijai pritaikytume aplinkos postūmį, naudotą tabu algoritmo su padidinta diversifikacijos strategija, ar jis taip pat suteiktų tokį stiprų agresyvios paieškos efektą? Ar teisinga taip daryti?

Algoritmų efektyvumo tyrimas buvo remiamas algoritmų iteracijų skaičiaus atžvilgiu (tik dalinės paieškos ir atsitiktinio klaidžiojimo algoritmai rėmėsi laiko trukme tam, kad jų rezultatai galėtų būti lyginami su kitų algoritmų rezultatais), tačiau, jei visiems algoritmams nustatytumėm vienodą laiko trukmę, ar nepasikeistų algoritmų efektyvumas sprendinio kokybės atžvilgiu? Taip pat, jei problemos formatas būtų mažesnis nei 10 x 10, arba didesnis, ar algoritmai pateiktų tuos pačius ar bent jau panašius rezultatus?

Koks turėtų būti hibridinis genetinis/tabu paieškos algoritmas tvarkaraščių sudarymui? Kokius jų komponentus galima hibridizuoti ir kuris turėtų būti bazinis algoritmas?

Jeigu įverčio funkcija neįtrauktų visų apribojimų, ar keistųsi algoritmų efektyvumas kiekvienai atskirai ar visoms kartu gamybinių tvarkaraščių klasėms?

# 8. LITERATŪRA


1. Blonskis, J., V. Bukšnaitis, A. Misevičius, D. Rubliauskas. C++ Builder grafika. Smaltija, Kaunas, 2004, 128 p.
2. Blonskis, J., V. Bukšnaitis, V. Jusas, R. Marcinkevičius, J. Smolinskas. Programavimo C++ Builder pavyzdžiai. Smaltija, Kaunas, 2002, 291 p.
3. Blum, C., Roli, A. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison / *ACM Computing Surveys*, Vol. 35, No. 3, September 2003, p. 268-308.
4. Čekanavičius V., G. Murauskas. Statistika ir jos taikymai. I. TEV, Vilnius, 2006, 240 p.
5. Choi, C. W. An Overview of Solving Job Shop Scheduling Problem by Local Search Techniques. July 19, 2000. http://citeseer.ist.psu.edu/424201.html, 5/22/2008.
6. Gendreau, M. An Introduction to Tabu Search. July 2002. http://www.ifi.uio.no/infheur/Bakgrunn/Intro_to_TS_Gendreau.htm, 5/22/2008.
7. Glover, F. Future paths and integer programming and links to artificial intelligence / *Comput. & Ops. Res*. Vol. 13, No. 5, 1986, p. 533-549.
8. Glover, F. Tabu Search: A Tutorial / *Interfaces* 20:4, July-August 1990, p. 74-94.
9. Glover, F. Tabu Search – Part I / *ORSA Journal on Computing*, Vol. 1, No. 3, Summer 1989, p. 190-206.
10. Glover, F. Tabu Search – Part II / *ORSA Journal on Computing*, Vol. 2, No. 1, winter 1990, p. 4-32.
11. Glover, F., M. Laguna. Tabu Search. Kluwer Academic Publishers, Norwell, 2001, 382 p.
12. Goldberg, D. E. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989, 372 p.
13. Hamming distance. http://en.wikipedia.org/wiki/Hamming_distance, 6/04/2008.
14. Hansen P., B. Jaumard. Algorithms for maximum satisfiability problem. RUTCOR Research Report No. 43-87, RUTCOR, Rutgers University, USA, 1987.
15. Haupt, R. L., S. E. Haupt. Practical Genetic Algorithms. Wiley-Interscience, New Jersey, 2004, p. 1-25, 187-203.
16. Hertz, A., E. Taillard, D. de Werra. A Tutorial on Tabu Search. http://www.cs.colostate.edu/~whitley/CS640/hertz92tutorial.pdf, 5/22/2008.
17. Holland, J. H. Adaptation in natural and artificial systems. The University of Michigan Press, 1975, Ann Harbor.
18. Listopadskis, N., E. Šakurovas. Genetinio algoritmo taikymas ir parametrų nustatymo problemos gamybinių tvarkaraščių sudarymui / Matematika ir matematinis modeliavimas / Kauno technologijos universitetas. ISSN 1822-2757. Kaunas : Technologija. T. 3 (2007), p. 164-171.

19. Martí, R. Multi-Start Methods / *Handbook of Metaheuristics*, (F. Glover, G.A. Kochenberg, eds.), Kluwer Academic Publishers, New York, 2003, p. 355-368.
20. Misevičius, A. Intelektualieji optimizavimo metodai / *Informacijos mokslai*, 2003, t. 26, ISSN 1392-0561, p. 160-166.
21. Misevičius, A., T. Blažauskas, J. Blonskis, J. Smolinskas. An Overview of Some Heuristic algorithms for Combinatorial Optimization Problems / *Informacinės technologijos ir valdymas*, ISSN 1392-124X, 2004, t. 30, Nr. 1, p. 21-31.
22. Misevičius, A., J. Blonskis. Experiments with Tabu Search for Random Quadratic Assignment Problems / *Information Technology and Control*, 2005, Vol. 34, No. 3, ISSN 1392-124X, p. 237-244.
23. Misevičius, A., J. Blonskis, V. Bukšnaitis. Kombinatorinis optimizavimas ir metaeuristiniai metodai: teoriniai aspektai / *Informacijos mokslai*, 2007, t. 42-43, ISSN 1392-0561, p. 213-219.
24. Misevičius, A., J. Blonskis, V. Bukšnaitis. Tabu paieška – efektyvus metodas kombinatorinio optimizavimo uždaviniams spręsti / *Informacijos mokslai*, 2004, t. 29, ISSN 1392-0561, p. 124-131.
25. Misevičius A., V. Bukšnaitis, J. Blonskis. Euristiniai algoritmai: tikslai, iššūkiai, metodologija, perspektyvos / *Informacijos mokslai*, 2006, t. 39, ISSN 1392-0561, p. 103-112.
26. Mitchell, M. An Introduction to Genetic Algorithms. A Bradford Book The MIT Press, Cambridge, 1999, p. 2-26.
27. Mockus J., A Set of Examples of Global and Discrete Optimization 2, Kluwer Academic Publishers, Boston/London/Dordrecht. http://mockus.us/optimum/docj/stud2.pdf, 6/04/2008.
28. Mockus J., W. Eddy, A. Mockus, L. Mockus, G. Reklaitis. Bayesian Heuristic Approach to Discrete and Global Optimization, Kluwer Academic Publishers, Boston/London/Dordrecht. http://mockus.us/optimum/docj/book.pdf, 6/04/2008.
29. Morrison S. Tabu Search for Process Scheduling. 1997. http://futurebatch.com/ ScheduleOptimization/ TabuSearch.pdf, 5/22/2008.
30. No Free Lunch Theorems. http://www.no-free-lunch.org, 5/22/2008.
31. Van Otterloo S. Evolutionary Algorithms and Scheduling Problems. August 7, 2002. http://citeseer.ist.psu.edu/650911.html, 5/22/2008.
32. Pham, D. T., Karaboga, D. Intelligent Optimisation Techniques: Genetic Algorithms, Tabu search, Simulated Annealing and Neural Networks. Springer, London, 2000, 302 p.
33. Reeves C. Genetic algorithms / *Handbook of Metaheuristics*, (F. Glover, G.A. Kochenberg, eds.), Kluwer Academic Publishers, New York, 2003, p. 55-82.
34. Sakalauskas L., G. Felinskas. Tvarkaraščių su ribotai ištekliais sudarymo euristiniai algoritmai, jų tyrimas ir taikymai / *Informacijos mokslai*, 2006, t. 38, ISSN 1392-0561, p. 90-103.

35. Šakurovas E., N. Listopadskis, Genetinio algoritmo taikymas ir parametrų nustatymo problemos gamybinių tvarkaraščių sudarymui / *Lietuvos matematikos rinkinys*, ISSN 0132-2818, Vilnius, 2007, t. 47, spec. nr., p. 479-483.
36. Taikomoji diskrečioji matematika; vadovėlis / K. Plukas, E. Mačikėnas, B. Jarašiūnaitė, I. Mikuckienė. K: Technologija, 2004, p. 248-249.
37. Taillard É. Scheduling instances. http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html, 5/22/2008.
38. Vaessens, R.J.M., E.H.L. Aarts, J.K. Lenstra. Job Shop Scheduling by Local Search. http://citeseer.ist.psu.edu/vaessens94job.html, 22/5/2008.

# 9. 1 PRIEDAS: TERMINŲ ŽODYNĖLIS

Kai kurie šiame darbe naudojami techniniai bei specifiniai terminai nėra galutinai nusistovėję lietuvių kalboje. Todėl šiame priede juos pateikiame originalo (anglų) kalba.

| *Terminas (anglų k.)* | *Vertimas (lietuvių k.)* |
|---|---|
| Ant Colony Optimization | Skruzdžių kolonijos optimizavimas |
| Flow Shop | Srautinis fabrikas |
| Job Shop | Darbų fabrikas |
| Open Shop | Atvirasis fabrikas |
| Quadratic Assignment Problem | Kvadratinio paskirstymo problema |
| Simulated Annealing | Atkaitinimo modeliavimas |
| Tabu Search | Tabu paieška |
| Traveling Salesman Problem | Keliaujančio komivajožieriaus problema |