KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
MULTIMEDIJOS INŽINERIJOS KATEDRA

Ieva Eimontienė

Iteratyvioji tabu paieška ir jos modifikacijos komivojažieriaus uždaviniui

Magistro darbas

Darbo vadovas
doc., dr. Alfonsas Misevičius

Kaunas
2007

TURINYS

PRATARMĖ

Viena iš aktualių informatikos krypčių yra optimizavimo algoritmai ir metodai, jų kūrimas ir tyrimas. Svarbią optimizavimo algoritmų grupę sudaro euristiniai algoritmai (EA). Euristiniai algoritmai bei metodai labai intensyviai taikomi sprendžiant kombinatorinio optimizavimo (KO) uždavinius, ir yra daugelio optimizavimo specialistų tyrinėjimų objektas.

Šio magistro darbo tematika yra viena iš euristinių algoritmų – iteratyviosios tabu paieškos (ITP) (angl. *iterated tabu search*) – sudarymas ir tyrimas. ITP metodas šiame darbe išbandytas sudėtingam kombinatorinio optimizavimo uždaviniui – komivojažieriaus uždaviniui (KU). KU priklauso NP-sunkių optimizavimo uždavinių klasei. Jau daugelį metų šis uždavinys laikomas savotiška „bandymų baze" kuriant ir tiriant algoritmus.

Vienas iš efektyvių euristinių metodų sprendžiant KU ir kitus KO uždavinius yra tabu paieška (TP) (angl. t*abu search*). Šiame darbe nagrinėjamas patobulintas tabu paieškos metodas, žinomas kaip iteratyvioji tabu paieška. Pasiūlytos kai kurios ITP metodo modifikacijos, besiremiančios tam tikromis sprendinių mutavimo (pertvarkymo) procedūromis (inversijos, įterpimai ir kt.), kurios įgalina pagerinti gaunamų sprendinių kokybę. Atlikti išsamūs sudaryto ITP algoritmo ir kitų pasiūlytų modifikacijų eksperimentiniai tyrimai, panaudojant testinius KU pavyzdžius iš KU testinių pavyzdžių bibliotekos TSPLIB (angl. *Travelling Salesman Problem Library*). Gauti rezultatai patvirtina pasiūlytų modifikacijų pranašumą kitų ITP variantų atžvilgiu.

Darbo struktūra yra tokia. Įvade (1-ame skyriuje) suformuluota užduotis, pagrindiniai darbo tikslai bei komivojažieriaus uždavinio matematinė formuluotė. 2-ame skyriuje apibūdinamos pagrindinės komivojažieriaus (KU) uždavinio sąvokos. 3-ame skyriuje aprašomi KU pritaikymai. Esamų kombinatorinių ir euristinių optimizavimų algoritmų situacija aprašoma 4-ame skyriuje. 5-ame skyriuje aprašomos naujosios algoritmo modifikacijos ir patobulinimai uždavinio sprendimui. Eksperimentinių bandymų rezultatai ir jų analizė pateikti 6-ame skyriuje. Darbas baigiamas išvadomis, literatūros sąrašu, terminų bei santraukų žodynu ir anotacijos santrauka anglų kalba. Papildomai pateikiami kai kurie priedai.

1. ĮVADAS

Darbo užduotis - programiškai realizuoti ir ištestuoti euristinį iteratyviosios tabu paieškos algoritmą, gerai žinomam kombinatorinio optimizavimo uždaviniui, komivojažieriaus uždaviniui. Pagrindinis dėmesys skirtas komivojažieriaus uždavinio sprendinių pertvarkymams bei iteratyviosios tabu paieškos algoritmo tyrimams. Darbo tikslai - ištirti iteratyviosios tabu paieškos algoritmą bei komivojažieriaus uždavinio sprendinių pertvarkymų poveikį šiam algoritmui.

Komivojažieriaus (dar vadinamas keliaujančio pirklio) uždavinys yra aplankyti visus savo maršruto miestus, tačiau tik po vieną kartą, pradedant ir baigiant kelionę savo mieste. Pagrindinis tikslas – rasti trumpiausią kelionės maršrutą.

Matematiškai komivojažieriaus (KU) uždavinys (angl. *traveling salesman problem*) [10, 11, 24] formuluojamas taip: duota atstumų tarp tam tikrų objektų (KU juos priimta vadinti miestais) matrica $\boldsymbol{D} = (d_{ij})_{n\times n}$ bei aibė Π, kurią sudaro visi galimi natūrinių skaičių nuo 1 iki n perstatymai. Tikslas yra surasti perstatymą $p_{\text{opt}} \in \Pi$, ir tokį, kad $p_{\text{opt}} = \arg\min_{p\in\Pi} z(p)$, čia z yra KU tikslo funkcija:

$$z(p) = \sum_{i=1}^{n-1} d_{p(i)p(i+1)} + d_{p(n)p(1)}; \qquad (1)$$

čia p atlieka KU sprendinio vaidmenį; n yra uždavinio apimtis. Perstatymui $p = (p(1), p(2), ..., p(n))$ ($p(i) \in \{1, 2, ..., n\}$) komivojažieriaus uždavinyje atitinka n miestų seka, kitaip tariant, maršrutas. Šiuo atveju $j = p(i)$ tiesiog žymi i-tąjį aplankytą maršruto miestą j. Perstatymo narių poros $(p(1), p(2))$, ..., $(p(i), p(i + 1))$, ..., $(p(n), p(1))$ paprastai vadinamos maršruto „atkarpomis“.

KU priklauso NP-sunkių uždavinių klasei [10], todėl norint išspręsti uždavinį, reikėtų įvertinti kiekvieną maršrutą, o tai reiškia, kad reikia $n!$ galimybių. Tokie uždaviniai tiksliai išsprendžiami tik esant ribotoms jų apimtims. Todėl vidutinės ir didelės apimties KU uždaviniams spręsti naudojami euristiniai algoritmai [23]. Reikšmingą tokių algoritmų grupę sudaro atkaitinimo modeliavimo (angl. *simulated annealing*) [2], tabu paieškos (angl. *tabu search*) [14], genetiniai (angl. *genetic algorithms*) [9] ir iteratyviosios lokaliosios paieškos (ILP) (angl. *iterated local search)* algoritmai. Vienas iš iteratyviosios tabu paieškos algoritmų variantų tiriamas magistro darbe.

2. BAZINĖS KOMIVOJAŽIERIAUS UŽDAVINIO SĄVOKOS

KU priklauso plačiai kombinatorinio optimizavimo [14] uždavinių klasei, kurio atveju sprendiniai yra perstatymai, o tikslo funkcijos vaidmenį vaidina funkcija z, aprašoma (1) formule. Komivojažieriaus uždaviniui galima „sukonstruoti“ įvairių aplinkos funkcijų. Dažnai naudojama vadinamoji „porinių sukeitimų“ funkcija Θ_2, kuri apibrėžiama tokia formule:

$$\Theta_2(p) = \{\, p' \mid p' \in \Pi,\ \rho(p,p') \le 2 \,\}; \tag{2}$$

čia p — bet kuris perstatymas iš Π, o $\rho(p,p')$ — atstumas tarp perstatymų p ir p'. Savaime suprantama, kad perstatymams netinka tolydžioms erdvėms įprastas Euklido atstumo tarp taškų apibrėžimas. Perstatymų atveju remiamasi vadinamojo Hamming‘o atstumo samprata. Natūralus būdas formaliai aprašyti šį atstumą tarp perstatymų p ir p' galėtų būti elementų, kurie yra skirtingose perstatymų p ir p' pozicijose, t.y. $\rho(p,p') = \left|\{i \mid p(i) \neq p'(i)\}\right|$, suskaičiavimas. Tačiau reikia pasakyti, kad KU atveju operuojama ne su atskirais perstatymų elementais $(j = p(i))$, o su perstatymų elementų poromis, t.y. „atkarpomis“ ($j_1 = p(i)$, $j_2 = p(i+1)$) (1 pav.). Turint tai omenyje, atstumo apibrėžimą tikslinga pakoreguoti: atstumas tarp dviejų perstatymų p ir p' lygus skaičiui elementų porų, esančių viename iš perstatymų (pvz. p), bet nesančių kitame (pvz. p') [9]. Matematinis atstumo tarp perstatymų p ir p' aprašymas šiuo atveju yra toks:

$$\rho(p,p') = |\Omega|; \tag{3}$$

čia aibė Ω yra sudaryta iš visų galimų elementų porų $(p(i), p((i \bmod n)+1))$ $(i \in \{1, 2, ..., n\})$, ir tokių, kad $\exists\, j$, ir toks, jog

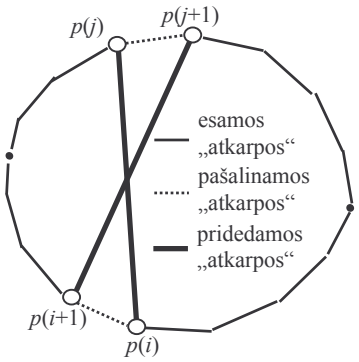
$$(p(i), p((i \bmod n)+1)) = \begin{cases} (p'(j), p'(j+1)), 1 \le j < n \\ (p'(j), p'(1)), \ j = n \end{cases} \tag{4a}$$

arba

$$(p(i), p((i \bmod n)+1)) = \begin{cases} (p'(j), p'(j-1)), 1 < j \le n \\ (p'(j), p'(n)), \ j = 1 \end{cases}. \tag{4b}$$

Sprendinio p aplinkos $\Theta_2(p)$ atskirą sprendinį p' galima pasiekti, atlikus mažiau ar daugiau sudėtingą sprendinio p transformaciją (pertvarkymą). KU sprendinių transformacijas, tiksliau „perėjimus“ iš duoto sprendinio į „sprendinį-kaimyną“ aprašysime vadinamojo dviejų „atkarpų“ sukeitimo operatoriaus $\phi(p,i,j)$ pagalba. Operatorius $\phi(p,i,j)$: $\Pi \times \mathrm{N} \times \mathrm{N} \to \Pi$ (čia N yra natūrinių skaičių aibė) duotą perstatymą p transformuoja į kitą perstatymą p', ir taip, kad

$\rho(p, p') = 2$; be to, perstatyme p yra pašalinamos elementų poros, esančios būtent i-oje ir j-oje pozicijose, t.y. poros $(p(i), p(i{+}1))$ ir $(p(j), p((j \bmod n){+}1))$; vietoje šių pridedamos (įterpiamos) naujos poros $(p(i), p(j))$ ir $(p(i{+}1), p((j \bmod n){+}1))$ (žr. 1 pav.). Jeigu dar tiksliau, operatorius $\phi(p,i,j)$ suformuoja perstatymą p', ir tokį, jog $p'(i) = p(i)$, $p'(i{+}1) = p(j)$, $p'(j) = p(i{+}1)$, $p'((j \bmod n){+}1) = p((j \bmod n){+}1)$, kai $1 \le i, j \le n \wedge 1 < j{-}i < n{-}1$; be to, jeigu $j{-}i{-}2 \ge 1$, tai $p'(i{+}k{+}1) = p(j{-}k)$ kiekvienam $k \in \{1, ..., j{-}i{-}2\}$ (tai būtina sąlygos $\rho(p, p') = 2$ galiojimo užtikrinimui). Toliau taip pat bus naudojama kompaktiška operatoriaus ϕ forma ϕ_{ij} perėjimui iš nesvarbu kurio perstatymo į perstatymą $\phi(\cdot,i,j)$. Tokiu būdu, pvz. užrašas $p' = p \oplus \phi_{ij}$ reikštų, kad p' yra gautas iš p panaudojant $\phi(p,i,j)$.



1 pav. Pašalinamos ir pridedamos „atkarpos“

Dviejų „atkarpų“ sukeitimo atveju labai nesudėtinga apskaičiuoti tikslo funkcijos (maršruto ilgio) z pokytį Δz, kuris gaunamas iš naujo maršruto p' ($p' = \phi(p,i,j)$) ilgio $z(p')$ atėmus prieš tai buvusio (esamo) maršruto p ilgį $z(p)$, t.y.

$$\Delta z = d_{p(i),p(j)} + d_{p(i+1),p((j \bmod n)+1)} - d_{p(i),p(i+1)} - d_{p(j),p((j \bmod n)+1)}. \qquad (5)$$

3. UŽDAVINIO TAIKYMAI

Komivojažieriaus uždavinys yra lengvai apibūdinamas, tačiau sunkiai sprendžiamas, kai susiduriama su didelėmis uždavinio apimtimis. Uždavinys yra ypatingai aktualus, kadangi pasižymi įvairiais praktiniais taikymais. Iš jų labiausiai paminėtini yra šie:

- krovinių gabenimo maršruto optimizavimas (minimizavimas);
- siuntinių pristatymas (laiškanešių maršrutai).

Optimalus maršrutas tarp 13509 JAV miestų pavyzdys parodytas 2 pav. [6].

Taip pat aktualūs ir kiti taikymai:

- vaikų autobusų maršutų nustatymas;
- spausdintų montažo plokščių (angl. *printed circuit board*) projektavimas;
- kristalų struktūros tyrimas;
- staklių darbo grafikas;
- eksponatų išdėstymas parodoje;
- sandėliavimas;
- garo turbinos rekonstravimas;
- transporto priemonių sustatymo aikštelėje tvarka;
- kompiuterių (telekomunikacijos, elektros) tinklų sudarymas;
- įvairiarūšių spinduliavimo žemėlapių konstravimas;
- tiriamų grandinių optimizavimas jungiant į vieną schemą;
- DNR nuoseklumo genetikoje vertinimas;
- komunikacijų, elektros ir telefonų laidų projektavimas;
- didelių duomenų kiekių kaupimas;
- mašinos darbų eiliškumo nustatymas ir pan. [6]

2 pav. Optimalus maršrutas tarp 13509 JAV miestų.

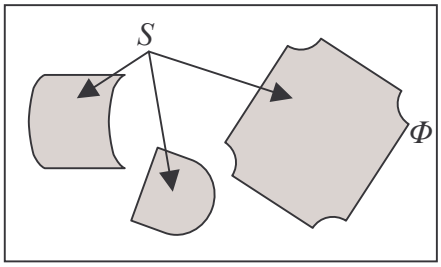
4. EURISTINIŲ ALGORITMŲ APŽVALGA

Dažnai realius optimizavimo uždavinius yra sunku suformuluoti ir išspręsti. Sunkumai iškyla ne tik sudarant uždavinio sprendimui reikalingą modelį, bet ir ieškant optimalių uždavinio parametrų reikšmių. Kartais tai reikalauja labai didelių skaičiavimų. Informatikos, vadybos, ekonomikos bei kitų sričių specialistai, sprendžiantys šiuolaikines problemas, dažnai susiduria su sudėtingais optimizavimo uždaviniais, kurių sprendimas reikalauja kiek gilesnio, negu vien metodo idėja, optimizavimo teorijos suvokimo. Todėl kiekvienam uždavinio sprendimui yra svarbu kuo geriau suprasti uždavinio sprendimui parinkto metodo teoriją ir jo efektyvumą.

4.1 Kombinatorinis optimizavimas – euristinių algoritmų taikymo sritis

Optimizavimo uždavinių tikslas yra surasti "geriausią" nepriklausomų dydžių (kintamųjų, reikšmių) konfigūraciją. Optimizavimo uždavinio tipą nusako šių kintamųjų įgyjamų reikšmių prigimtis. Kintamųjų įgyjamos reikšmėmis gali būti skaičiai, skaičių rinkiniai (vektoriai), grafai, poaibiai, perstatymai ir kiti objektai. Pagal jų įvairovę, galima išskirti dvi optimizavimo uždavinių kintamųjų tipų grupes: uždaviniai, kuriuose kintamųjų įgyjamos reikšmės yra tolydžios (pvz. reikšmės iš realiųjų skaičių aibės) ir uždaviniai, kuriuose kintamųjų įgyjamos reikšmės yra tik diskretinės (pvz. reikšmės iš natūrinių, sveikųjų skaičių ar kurios kitos baigtinės aibės). Optimizavimo uždaviniai, kuriuose operuojama būtent su diskretinio tipo kintamųjų reikšmėmis, o kintamųjų įgyjamų reikšmių aibė, t.y. aibė, kurioje ieškoma „geriausių" kintamųjų reikšmių, yra baigtinė ar bent jau suskaičiuojama, – vadinami kombinatorinio optimizavimo (KO) uždaviniais [23].

Kombinatorinio optimizavimo uždavinį galima aprašyti poros (S, f) pagalba, kur S yra leistinųjų sprendinių aibė. Bendruoju atveju aibė S yra vadinamosios bazinės aibės Φ poaibis $(S \subseteq \Phi)$. Φ yra visų galimų reikšmių, kurias gali įgyti optimizavimo uždavinio kintamieji, aibė. (Pavyzdžiui, jeigu $X=\{x_1, x_2, ..., x_m\}$ yra KO uždavinio kintamųjų aibė, tai Φ galėtume apibrėžti taip: $\Phi=D_1\times D_2\times...\times D_m$, čia $D_1, D_2, ..., D_m$ yra kintamųjų, atitinkamai $x_1, x_2, ..., x_m$, galimų įgyti reikšmių aibės (domenai).) Aibę S sudaro tie aibės Φ sprendiniai, kurie tenkina tam tikras sąlygas (apribojimus), t.y. $S=\{s|s \in \Phi, s$ tenkina keliamas sąlygas$\}$. Beje, aibė S gali būti sudaryta iš didesnio ar mažesnio „nepersidengiančių" poaibių skaičiaus (žr. 3 pav.). Aibės S ir Φ gali sutapti – šiuo atveju visi sprendiniai yra leistinieji $(S \equiv \Phi)$.



3 pav. Bazinė aibė Φ ir leistinųjų sprendinių aibė S

Nemažą KO uždavinių klasę sudaro uždaviniai, kuriuose kintamųjų reikšmių tipas yra perstatymas. Perstatymas, kaip žinoma, yra kurios nors baigtinės aibės nesikartojančių elementų seka. Tokios aibės vaidmenyje gali būti aibė, sudaryta iš natūrinių skaičių, tarsime nuo *1* iki *n*. Šiuo atveju leistinųjų sprendinių aibę *S*, kuri tuo pačiu yra bazinė aibė (t.y. $S \equiv \Phi$), sudaro visi įmanomi natūrinių skaičių nuo *1* iki *n* perstatymai, t.y.

$$S=\{s|s=(s(1), s(2),...,s(n)),\ s(i)\in\{1,2,...,n\}\ \forall\ 1\le i\le n,\ s(i)\ne s(j)\ \forall\ i,j\ 1\le i,j\le n,\ i\ne j\};\qquad (6)$$

čia *s* žymi perstatymą (sprendinį-perstatymą), o *s(i)* – perstatymo *s* i-tąjį elementą (narį); *n* yra laikoma uždavinio apimtimi[23].

Optimizavimo uždaviniui pilnai apibrėžti turi būti nurodomas tam tikras sprendinių vertės matas, kiekvienam (leistinam) sprendiniui įgaunantis kurią nors (realią) skaitinę reikšmę. Kitaip tariant, turi būti apibrėžta funkcija *f* (tai yra antrasis poros *(S, f)* narys), kurios apibrėžimo sritis yra aibė *S*, o reikšmių aibė – pvz. realių skaičių aibė R^1 (trumpai žymima f: $S \to R^1$). Ši funkcija vadinama optimizavimo uždavinio tikslo funkcija (sprendinių optimalumo kriterijumi). Toliau, neprarasdami universalumo, tarsime, jog tikslo funkcija *f* turi būti minimizuojama (iš tiesų, funkcijos *f* maksimizavimas yra ekvivalentiškas funkcijos *–f* minimizavimui).

Išspręsti kombinatorinio optimizavimo uždavinį *(S, f)* reiškia, kad reikia tarp visų aibės *S* sprendinių $s_1, s_2, ..., s_k, ..., s_{|S|}$ surasti tokį sprendinį, kuriam tikslo funkcijos *f* reikšmė būtų lygi mažiausiai galimai; t.y. turi būti gautas sprendinys $s_{opt} \in S$, toks, jog:

$$S_{opt} \in S_{opt} = \left\{ s^{\nabla} \mid s^{\nabla} = \arg\min_{s\in S} f(s) \right\} \qquad (7)$$

Sprendinys s_{opt}, kuris minimizuoja tikslo funkciją *f*, vadinama uždavinio *(S, f)* (globaliai) optimaliu sprendiniu (globaliuoju optimumu arba tiesiog optimumu), aibė $S_{opt} \subseteq S$ – (globaliai) optimalių sprendinių aibe, o reikšmė $f_{opt} = f(s_{opt})$ – optimalia tikslo funkcijos reikšme.

Aibės *S* sprendiniams galima apibrėžti atstumo matą. Atstumą charakterizuoja funkcija $\rho : S \times S \to R^1$, bet kuriai sprendinių porai *(s, s′)* priskiriančia tam tikrą neneigiamą skaitinę reikšmę. Esant sprendiniams-perstatymams, atstumas tarp sprendinių

dažnai apibrėžiamas vadinamosios Hamming‘o funkcijos pagalba. Štai Hamming‘o atstumo apibrėžimo pavyzdys. Tarkime, kad s ir s' yra sprendinių-perstatymų pora iš S. Vienas iš natūralių būdų apibrėžti Hamming‘o atstumą tarp s ir s' yra suskaičiuoti elementus, kurie yra skirtingose perstatymų s ir s' pozicijose, t.y.

$$\rho(s,s') = |\{i \mid s(i) \neq s'(i)\}|. \quad (8)$$

Toliau apibrėšime vadinamąją aplinkos funkciją $\Theta : S \to 2^S$ (čia 2^S žymi aibės S visų galimų poaibių aibę). Šios funkcijos pagalba bet kuriam sprendiniui $s \in S$ priskiriama aibė $\Theta(s) \subseteq S$ - sprendinio s aplinka, dar vadinama „aplinkinių“ sprendinių (arba „sprendinių-kaimynų“) aibe. Konkrečios aplinkos pavyzdys yra vadinamoji „k-aplinka“ (ją žymėsime Θ_k). Aplinka Θ_k apibrėžiama paprastai: sprendinio s „k-aplinką“ sudaro visi tie sprendiniai, kurie „nutolę“ nuo duoto sprendinio atstumu, ne didesniu kaip k, t.y.

$$\Theta_k(s) = \{s' \mid s' \in S, \rho(s,s') \leq k\}; \quad (9)$$

čia s yra bet kuris sprendinys iš S, ρ - Hamming‘o atstumas tarp sprendinių, o k gali būti interpretuojamas ir kaip savotiškas aplinkos „radiusas“.

Prieš pradėdami kalbėti apie KO uždavinių sprendinio algoritmus, dar apibrėšime lokaliai optimalaus sprendinio sąvoką. Pažymėkime $\Delta_{s,s'} = f(s') - f(s)$ tikslo funkcijos reikšmių pokytį, gautą „perėjus“ iš sprendinio s į sprendinį s'. Sprendinys s yra vadinamas lokaliai optimaliu aplinkos Θ atžvilgiu, jeigu kiekvienam sprendiniui s' iš aplinkos $\Theta(s)$ galioja $\Delta_{s,s'} \geq 0$ (visi tikslo funkcijos pokyčiai yra teigiami); kitaip tariant, sprendinio s aplinkoje Θ nėra nei vieno geresnio sprendinio už s [23].

4.2 Modernieji euristiniai algoritmai

Šiame skyriuje iš pradžių aptariame kai kuriuos algoritmų klasifikavimo klausimus. Po to pateikiama bendroji algoritmų veikimo schema (paradigma) ir apžvelgiami modernieji euristiniai metodai bei algoritmai.

4.2.1 Euristinių algoritmų klasifikavimas

Algoritmų kombinatorinio optimizavimo uždaviniams spręsti visuma yra skirstoma į dideles savarankiškas kategorijas:

- Euristiniai algoritmai
  - Klasikiniai LP algoritmai
  - Tabu paieška
  - Iteratyvioji lokalioji paieška

- Hibridiniai algoritmai
- Atkaitinimo modeliavimas
- Genetiniai algoritmai
- Kiti algoritmai

- Apytiksliai algoritmai
- Tikslieji algoritmai

Euristiniai algoritmai (EA) (angl. h*euristic algorithms*) [23] yra tokie optimizavimo uždavinių sprendimo metodai, kuriais siekiama surasti aukštos kokybės, bet nebūtinai optimalius per priimtiną laiką. EA sprendžiamiems uždaviniams paprastai suranda tik mažiau ar daugiau nuo optimumo nutolusius sprendinius – lokaliai optimalius sprendinius. Todėl šie algoritmai literatūroje dar dažnai vadinami lokaliosios paieškos (LP) algoritmais arba tiesiog lokaliąja paieška (angl. *local search*).

Euristiniai algoritmai neužtikrina sprendinio optimalumo ir nėra garantuojama, kad surastas sprendinys bus "nutolęs" nuo optimumo ne daugiau kaip tam tikras fiksuotas atstumas. Tuo EA skiriasi nuo apytikslių algoritmų, kurie užtikrina, kad gautas sprendinys yra "nutolęs" nuo optimumo ne daugiau kaip iš anksto duotas atstumas $\varepsilon>0$. Apytiksliai algoritmai užima savotišką tarpinę padėtį tarp euristinių ir tiksliųjų algoritmų. Pastarieji garantuoja optimalaus sprendinio (globaliojo optimumo) suradimą, t.y. uždavinio išsprendimą. Tiesa, laikas, reikalingas optimumui pasiekti, dažnai yra susietas su uždavinio apimtimi eksponentine priklausomybe. Tuo tarpu, EA pasižymi polinominiu paieškos laiko priklausomybės nuo uždavinio apimties pobūdžiu, tačiau šiuo atveju prarandama optimumo suradimo garantija – sprendinių tikslumas "aukojamas" vardan paieškos laiko.

Nagrinėjant euristinius algoritmus susiduriama su šios kategorijos atstovų įvairove ir daugialypiškumu. Euristinius algoritmus galime klasifikuoti atsižvelgiant į įvairius būdingus požymius, pvz. pagal tai, kokie procesai ir kokiu abstrahavimo lygiu modeliuojami, koks algoritmo intelektualizavimo, sudėtingumo laipsnis, kiek ir kokių yra algoritmo struktūrinių dalių, komponentų, kokios kokybės sprendinių paieškai orientuotas algoritmas ir t.t. Šių algoritmų spektras apima dešimtis, gal net šimtus paieškos strategijų ir jų variantų, pradedant elementariomis klasikinėmis "godžiosios" paieškos procedūromis ir baigiant labai jau intelektualizuotais, "save apmokančiais" metodais [23].

4.2.2 Euristinių algoritmų veikimo principas

Bendroji euristinių algoritmų veikimo principas (schema) galėtų būti aprašytas taip: paieška pradedama nuo tam tikru būdu sukonstruoto pradinio sprendinio; toliau paieškos

procesas vykdomas nuosekliai atliekant tam tikrus sprendinių pertvarkymus ir „pereinant“ nuo vieno sprendinio prie kito (atsižvelgiant į aplinkos funkciją); „perėjimai“ valdomi, kitaip tariant, sprendimai dėl „perėjimų“ priimami orientuojantis į sprendinių kokybę, t.y. tikslo funkcijos reikšmes; jei proceso eigoje gautas sprendinys pasirodo esąs geresnis negu iki tol rastas geriausias, tai tas sprendinys „įsimenamas“; procesas tęsiamas tol, kol nepatenkinama baigimo sąlyga; geriausias „įsimintas“ sprendinys laikomas paieškos rezultatu.

Formalūs euristinių algoritmų paradigmos žingsniai yra šie:

1. Sukonstruoti (sugeneruoti) pradinį sprendinį (arba sprendinių grupę).
2. Atsižvelgiant į aplinkos funkciją, atlikti vieną ar daugiau esamo sprendinio (ar sprendinių grupės) transformacijų gaunant naują(us) sprendinį(ius). (Esant reikalui, vietoje sprendinio(ių) transformavimo gali būti vykdomas naujo(ų) sprendinio(ių) konstravimas (generavimas).)
3. Atsižvelgiant į transformacijos funkcijos reikšmę(es) naujam(iems) sprendiniui(iams), padaryti sprendimą.
4. Jeigu sprendimas yra „teigiamas“, tai:
   (4a) esamą sprendinį (sprendinių grupę) pakeisti nauju sprendiniu (sprendinių grupe) ir naudoti kaip esamą sprendinį tolimesnėse iteracijose;
   (4b) jeigu naujas (ar vienas iš naujų) sprendinys yra geresnis už iki šiol geriausią rastą, tai „įsiminti“ tą sprendinį kaip geriausią; jeigu sprendimas yra „neigiamas“, tai tęsti paiešką nekeičiant esamo sprendinio.
5. Jeigu paieškos baigimo sąlyga nepatenkinta – kartoti (2)—(5) punktus; priešingu atveju — baigti.

4.2.3 Kai kurie modernieji euristiniai algoritmai

Bendras kombinatorinio optimizavimo uždavinių sprendimo principas yra toks: paieška pradedama nuo tam tikro pradinio sprendinio; toliau vykdomas sprendinio gerinimas, panaudojant kurį nors euristinį, t.y. lokaliosios paieškos metodą.

Pradinį sprendinį sudaryti galima atsitiktiniu būdu (pvz. generuojant n sveikų nepasikartojančių atsitiktinių skaičių iš intervalo $[1,n]$). Sprendinio konstravimui galima panaudoti kurį nors lokaliosios paieškos algoritmą. Paprastai konstruojant pradinį sprendinį, yra atsižvelgiama į tikslo funkcijos reikšmę. Ši reikšmė perskaičiuojama kiekvieną kartą, kai tik į dalinai sukonstruotą sprendinį įtraukiamas naujas komponentas. Dažniausiai yra naudojama godžia strategija pagrįstas algoritmas: į konstruojamą sprendinį įtraukiamas toks

komponentas, kad tikslo funkcijos f reikšmė, apskaičiuota dalinai sukonstruotam sprendiniui, būtų minimali. Taip kartojama tol, kol visi komponentai įtraukiami į sprendinį.

Pradinio sprendinio konstravimo algoritmai neužtikrina pakankamos gauto sprendinio kokybės, todėl reikalingas sprendinio gerinimas ir atitinkami algoritmai [4].

Atkaitinimo modeliavimas

Atkaitinimo modeliavimo (AM) metodo ištakos slypi statistinėje mechanikoje, o jeigu tiksliau, energetinių procesų, vykstančių sistemose, sudarytose iš didelio skaičiaus dalelių, imitavime. Šio imitavimo esmė slypi tame, jog iš pradžių sistema „pervedama“ į didelės energijos būseną, o po to, palaipsniui mažinant energiją, stengiamasi pasiekti būseną, atitinkančią žemiausią sistemos energetinį lygmenį – tarsi kūnas būtų įkaitintas iki pakankamai aukštos temperatūros, o paskui, jį atkaitinant, t.y., mažinant temperatūrą, jis būtų savotiškai „užgrūdinamas“. Šio metodo pradininkai (1953m.) buvo Metropolis, Rozenbluthai ir kt. [21]. Jie pasinaudojo Bolcmano (eksponentiniu) pasiskirstymo dėsniu apibrėždami tikimybę, kad sistema, įvykus joje tam tikrai perturbacijai (pasikeitimui), pereis iš vieno energijos lygio (E_1) į kitą (E_2), kai temperatūra lygi t: $P = \begin{cases} 1, \Delta E < 0 \\ e^{-\Delta E / C_{E^T}}, \Delta E \geq 0 \end{cases}$, čia $\Delta E = E_2 - E_1$, o C – konstanta. Cerný [5] ir Kirkpatrickas su bendraautoriais [18] buvo pirmieji, pritaikę atkaitinimo modeliavimo metodą sprendžiant kombinatorinio optimizavimo uždavinius.

Konkrečios atkaitinimo modeliavimo algoritmų realizacijos skiriasi viena nuo kitos šiais veiksniais (faktoriais): aplinkos funkcija, atkaitinimo („atšaldymo“) schema ir baigimo sąlyga. Viena iš dažniausiai naudojamų aplinkos funkcijų yra vadinamoji porinių sukeitimų funkcija N_2.

Atkaitinimo modeliavimo algoritmų veikimo principas yra gana paprastas: pradėti nuo atsitiktiniu (ar kitu) būdu sukonstruoto sprendinio; iš esamo sprendinio s aplinkos parinkti sprendinį s' ir apskaičiuoti tikslo funkcijos skirtumą $\Delta f = f(s') - f(s)$. Sprendimo taisyklė yra tokia: jeigu tikslo funkcijos reikšmė pagerėja ($\Delta f < 0$), tai esamą sprendinį s pakeisti nauju sprendiniu s' ir naudoti kaip išeities „tašką“ tolesniuose bandymuose; jeigu $\Delta f \geq 0$, tai daryti pakeitimą su tikimybe $\mathrm{P}(\Delta f) = \mathrm{e}^{-\Delta f / t}$ (čia t yra einamoji temperatūra). Procesas tęsiamas tol, kol patenkinama algoritmo baigimo sąlyga. Metodo algoritmo rezultatas yra geriausias surastas sprendinys, tačiau nebūtinai tas, kuris gautas paskutinėje paieškos iteracijoje – tas sprendinys „įsimenamas“ jo suradimo momentu.

Genetiniai algoritmai

Genetinių algoritmų (GA) ir jų sudarymo principų pradininku buvo Holland'as[15]. Genetinių algoritmų veikimas yra pagrįstas evoliucijos, vykstančios gyvojoje gamtoje imitavimu. Pagrindinės sąvokos, kurios naudojamos modeliuojant evoliucijos procesus, yra „individas" ir „populiacija". „Individas" yra tam tikras elementarus, daugiau neskaidomas vienetas. Didesnė ar mažesnė „individų" grupė sudaro „populiaciją". Svarbus dalykas yra ir vadinamasis „individo tinkamumas" – savotiška „individo" vertė. „Vertingesnis" (grynai biologiškai) yra tas „individas", kuris sugeba geriausiai prisitaikyti (būti „stipresnis" už kitus), pvz. palikti didesnį palikuonių skaičių.

Optimizavime vietoje sąvokų „individas", „populiacija", „individo tinkamumas" naudojamos įprastos sąvokos: „individui" atitinka atskiras sprendinys, „populiacijai" – sprendinių aibė (rinkinys), pagaliau, „individo vertė" yra asocijuojama su tikslo funkcijos reikšme duotajam sprendiniui. Taip, kaip gyvosios gamtos evoliucijoje išlieka tik stipriausi „individai", taip ir optimizavime GA pagalba siekiama gauti kuo geresnį sprendinį.

Terminas „genetiniai algoritmai" nusako tam tikrą, gana universalų meta-metodą – algoritmų su panašiomis savybėmis klasę (šeimą). Tos savybės yra tokios:

1. Operuojama su viena ar daugiau sprendinių „populiacijų".
2. Atrenkant tolimesniam apdorojimui atskirus sprendinius iš sprendinių „populiacijos", pirmenybė teikiama tiems sprendiniams, kurie turi geresnes tikslo funkcijos reikšmes.
3. Naudojamas tam tikras mechanizmas, kuris skirtas formuoti naujiems sprendiniams, kombinuojant („kryžminant") turimus sprendinius.
4. Esant reikalui, panaudojamas papildomas sprendinių randomizuoto pertvarkymo mechanizmas.
5. Tam tikrais periodais „populiacija" atnaujinama, pašalinant iš jos, pvz. blogesnius sprendinius ir paliekant joje geresnius bei naujai suformuotus sprendinius.

Formalizuojant genetinio algoritmo aprašymą, aukščiau nurodytos savybės, „mechanizmai" susiejami su atitinkamomis procedūromis. Taip antroji savybė susiejama su vadinamąja atrinkimo procedūra, trečiasis mechanizmas tradiciškai vadinamas krosoveriu, ketvirtasis mechanizmas – mutavimu. Penktajame žingsnyje atliekamas „populiacijos" atnaujinimas – tai savotiškas „gryninimas". Minėtosios procedūros turi būti kartojamos tam tikrą, pakankamai didelį skaičių kartų tam, kad genetinis algoritmas galėtų konverguoti į aukštos kokybės lokalųjį optimumą.

5. TYRIMO DALIS

5.1 Tabu paieškos metodas

F. Gloveris 1986 metais pasiūlė Tabu paieškos (TP) metodą [12]. Jis tapo labai populiarus ir sėkmingai taikomas įvairiems kombinatorinio optimizavimo uždaviniams, iš jų grafų dalijimo, grafų dažymo, komivojažieriaus, kvadratinio paskirstymo ir kitiems uždaviniams [11]. Ir šiomis dienomis TP metodas vis dar išlieka daugelio optimizavimo specialistų intensyvių tyrinėjimų objektu, nuolat pateikiamos naujos šio metodo versijos ir patobulinimai. Šis metodas remiasi išplėsta lokaliąja paieška. Įprasti klasikiniai paieškų algoritmai apsiriboja lokalaus optimalaus sprendinio suradimu nagrinėjamo sprendinio aplinkoje, o algoritmai, pagrįsti TP, tęsia paiešką net ir tuo atveju, kai surandamas lokaliai optimalus sprendinys, t.y. kada duotojo sprendinio aplinkoje neįmanoma surasti geresnio sprendinio. Tabu paieška yra iteracinis geresnių sprendinių paieškos procesas, kuris leidžia daugkartinį „perėjimą" nuo vieno lokalaus optimumo (LO), prie kito įsimenant geriausiąjį iš surastų sprendinių. Pagrindinė TP esmė yra atlikti perėjimus net ir tada, kai nėra pagerinamas kaimyninis sprendinys. Tačiau TP yra pagrįsta perėjimais, kurie yra draudžiami tam, kad būtų išvengta ciklinimų [22].

Tabu paieška pradedama nuo pradinio, galbūt atsitiktinai sugeneruoto, sprendinio s iš aibės S, po to iteraciniu būdu kartojamas perėjimų iš vieno sprendinio į kitą procesas. Kiekvienu procedūros žingsniu yra analizuojama einamojo sprendinio s kaimyninių sprendinių aibė $\Theta(s)$, ir atliekamas tas perėjimas, kuris labiausiai pagerina (sumažina) tikslo funkcijos f reikšmę. Jeigu nėra pagerinančių perėjimų, tai pasirenkamas tas, kuris mažiausiai pablogina (padidina) tikslo funkcijos f reikšmę. Trumpai tariant, perėjimas atliekamas į geriausiąjį kaimyninį sprendinį s' iš $\Theta(s)$.

Siekiant išvengti grįžimo į neseniai nagrinėtą sprendinį, atvirkštinis perėjimas turi būti draudžiamas. Tai realizuojama įsimenant šį perėjimą (arba perėjimo požymį) tam tikroje atmintyje, vadinamoje tabu sąrašu (T). Šiame sąraše įsimenama $h=|T|$ paskutiniųjų perėjimų „žymės" („pėdsakai"). Dydis $h=|T|$ yra vadinamas tabu sąrašo ilgiu. Šis ilgis yra labai svarbus: jeigu jis yra labai mažas, tai gali būti gaunamas nepageidaujamas ciklinimas; tuo tarpu, jeigu – labai didelis, tai apribojama paieška srityse, „teikiančiose vilčių". Taigi perėjimas iš sprendinio s į sprendinį $s' \in \Theta(s)$ yra traktuojamas kaip tabu, jeigu jis (ar jo požymis) yra sąraše T. Taip siekiama apsisaugoti nuo sugrįžimo į jau „aplankytus" sprendinius ir algoritmo „užsiciklinimo". Tačiau, po tam tikro laiko, grįžimas į anksčiau

„aplankytus“ sprendinius gali būti naudingas. Dėl tos priežasties, įvedamas vadinamasis aspiracijos kriterijus tam, kad leisti „tabu būsenai“ būti atšauktai esant tam tikroms palankioms aplinkybėms, pavyzdžiui, tabu perėjimą iš s į s' galima leisti, jeigu $f(s')<f(s^*)$, kur s^* yra geriausias iki šiol rastas sprendinys). Tiek tabu sąrašas, tiek aspiracijos kriterijus yra dinamiškai atnaujinami algoritmo vykdymo eigoje. Paprastai, procesas yra baigiamas, kai tik atliekamas iš anksto užduotas bandymų skaičius. Tabu paieškos paradigma pavaizduota 4 pav.

```
function tabu_paieška(s);
  // pradiniai duomenys: s – pradinis sprendinys; rezultatai: s* – geriausias rastas sprendinys //
  s* := s;
  inicializuoti tabu sąrašą T;
  repeat // vykdyti tabu paieškos ciklą //
    rasti geriausią sprendinį s′ ∈ Θ′(s) ⊆ Θ(s), čia Θ′(s) – sprendinio s aplinkos
    poaibis, kurio sprendiniai (arba "perėjimo" iš s į Θ(s) „pėdsakai")
    nepriklauso tabu sąrašui T arba tenkina aspiracijos sąlygą;
    s := s′; // pakeisti esamą sprendinį nauju ir naudoti kaip „išeities tašką“ tolimesnėse iteracijose //
    įsiminti sprendinį s (arba "perėjimo" iš s į s′ „pėdsaką") tabu sąraše T;
    if f(s) < f(s*) then s* := s; // įsiminti geriausią rastą sprendinį //
    jei reikia, atnaujinti tabu sąrašą T
  until patenkinta baigimo sąlyga;
  return s*
end.
```

4 pav. Standartinės tabu paieškos metodo schema

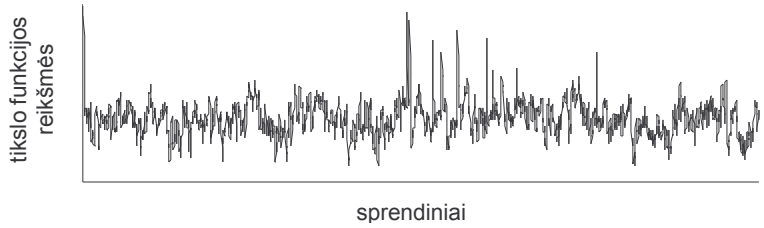
5.2 Iteratyvioji tabu paieška: bendrieji aspektai

Net ir panaudojus modernius euristinius metodus, pvz. tabu paiešką [11,12], vis dar išlieka rimtų sunkumų, ypač sprendžiant didesnės apimties uždavinius, arba kai reikia gauti sprendinius, labai artimus optimaliems. Svarbesni tuos sunkumus sąlygojantys faktoriai:

- milžiniškas lokaliųjų optimumų sprendinių „erdvėje“ kiekis;
- paieškos „trajektorijų“ ciklai;
- „determinuoto chaoso“ fenomenas.

Pastarąjį galima charakterizuoti kaip paieškos „trajektorijų“ užsklendimą atskirose sprendinių „erdvės“ dalyse [3]. Nors sprendinių aibė ir yra baigtinė (tai galima traktuoti kaip tam tikrą determinuotą sistemą), o paieškos „trajektorijos“ yra laikytinos atsitiktinėmis. Dėl šio „determinuoto chaoso“ fenomeno pasireiškimo gali būti labai sunku surasti globalųjį

optimumą (GO), ypač tais atvejais, kai uždavinio tikslo funkcija yra daugiaekstremė ir optimumas nepatenka į galimas „kolapsų“ sritis, pvz. GO tašką supa daugybė „gilių“ lokaliųjų optimumų „tarpeklių“ (žr. 5 pav.).



5 pav. Galimos tikslo funkcijos *„trajektorijos“* optimizavimo uždavinio sprendiniams fragmentas

Padėti eliminuoti ar bent jau sušvelninti šių negatyvių faktorių įtaką galėtų iteratyvioji tabu paieška (ITP). Iteratyvioji tabu paieška (ITP) yra viena iš naujausių euristinių algoritmų grupių. ITP savo ruožtu remiasi iteratyviosios lokaliosios paieškos (ILP) koncepcija [20]. Iteratyvioji lokalioji paieška [1] yra viena iš naujausių euristinių algoritmų metodų. ILP esmė glūdi tame, jog tradicinių EA rezultatus galima pagerinti panaudojant tam tikrus sprendinių pertvarkymus. Nesigilinant į detales, ILP gali būti lakoniškai įvardijama kaip „griauti ir atstatyti“ principu grįsta paieškos strategija. Tokios strategijos tikslas – „pereinant“ nuo vieno lokaliai optimalaus sprendinio prie kito, stengtis aprėpti kuo didesnį sprendinių „poerdvį“, taip sudarant palankesnes sąlygas artėti prie GO [24].

5.3 Iteratyviosios tabu paieškos algoritmo realizacija

Iteratyviajai tabu paieškai būdingi du pagrindiniai etapai (fazės):

1) sprendinio pagerinimas naudojant tabu paieškos algoritmą,
2) sprendinio rekonstravimas (mutavimas) naudojant tam tikslui skirtą procedūrą.

Tarp šių etapų paprastai įsiterpia ne toks svarbus sprendinio kandidato atrankos rekonstravimui žingsnis; be to, kai kuriais atvejais, kai ilgą laiką nepavyksta surasti geresnių sprendinių (t.y. pasireiškia savotiška paieškos „stagnacija“), vietoje rekonstravimo vykdomas grynai atsitiktinio sprendinio generavimas („šaltasis restartas“). ITP inicijuojama pradinio sprendinio pagerinimu, gaunant (pirmąjį) lokaliai optimalų sprendinį, tarkime $s^{\bullet}$. Gautasis lokaliai optimalus sprendinys yra tam tikru mastu „suardomas“, tiksliau, rekonstruojamas gaunant naują sprendinį, pavyzdžiui, $s^{\sim}$. Rekonstruojant sprendinį nesiekiama jo visiškai, absoliučiai „sugriauti“. Atvirkščiai, tikslinga, kad gautas naujas sprendinys „paveldėtų“ tam tikras geras ankstesnio sprendinio charakteristikas. Tačiau kartu turi būti užtikrinamas ir deramas diversifikavimo (įvairovės) lygis. Taigi rekonstravimas neturi būti nei per daug

„stiprus“, nei per daug „silpnas“. Pirmuoju atveju paieška taptų labai panaši į daugkartinę paiešką startuojant tiesiog nuo atsitiktinių sprendinių, ir būtų prarandama naudinga paieškos metu sukaupta informacija; antruoju atveju rekonstravimas negarantuotų pakankamai „nutolusio“ (nuo duotojo) sprendinio generavimo – o tai lemtų „grįžimą“ atgal į ankstesnįjį LO po sprendinio pagerinimo etapo. Rekonstruotas sprendinys $s^{\sim}$ vaidina pradinio sprendinio iš naujo startuojančiai TP procedūrai vaidmenį. TP procedūra grąžina naują lokaliai optimalų sprendinį $s^{\bullet}$, šis vėl rekonstruojamas, ir t.t. Geriausias lokalusis optimumas ($s*$) „įsimenamas“.

Sprendinių pagerinimo (TP būdu) ir rekonstravimo etapai iteraciniu būdu kartojami kiek norima daug kartų; paprastai ITP vykdymo trukmė valdoma iteracijų skaičiumi. Apibendrinta ITP schema pateikiama 6 paveiksle.

```
function iteratyvioji_tabu_paieška(s);
 // pradiniai duomenys: s – pradinis sprendinys;  rezultatai: s* – geriausias rastas sprendinys //
begin
  s*:=tabu_paieška(s);  //pradinio sprendinio pagerinimas panaudojant TP procedūrą //
  s:=s•; s•:=s*;
  repeat  // vykdomas iteratyviosios tabu paieškos ciklas //
    if ilgą laiko tarpą nerasta geresnio sprendinio then s~ := naujas sprendinys
    s:=kandidato_nustatymas(s, s•, ...);  // parenkamas sprendinys-kandidatas rekonstravimui //
    else begin
      s := kandidato_nustatymas(s, s•);// parinkti sprendinį rekonstravimui //
      s~:= rekonstravimas(s)  // rekonstruoti sprendinį s gaunant sprendinį s~ //
    end;
  until patenkinta baigimo sąlyga
  return s*
end.
```

6 pav. Iteratyviosios tabu paieškos paradigma

5.3.1 Pradinio sprendinio konstravimas

Egzistuoja keletas nuomonių apie tai, kaip būtų geriausia konstruoti pradinius sprendinius. Antai, Reinelt‘as [25] eksperimentuodamas priėjo išvados, kad yra geriau formuoti pradinius sprendinius panaudojant tam tikslui skirtą efektyvų konstrukcinį euristinį algoritmą, negu tiesiog generuoti visai atsitiktinius sprendinius. Tačiau vieno iš efektyviausių algoritmų KU autoriai Lin‘as ir Kernighan‘as [19] teigė, jog konstrukcinių — neretai gana sudėtingų — algoritmų naudojimas tėra tik „laiko švaistymas“: rezultatų kokybę lemia iš esmės iteracinis algoritmas. Be to, konstrukciniai euristiniai algoritmai dažnai yra savo

prigimtimi deterministiniai: duotas algoritmas tam pačiam duomenų rinkiniui sukuria tik vieną ir tą patį sprendinį; tuo tarpu pageidautina turėti galimybę kurti įvairius pradinius sprendinius. Iteratyviosios tabu paieškos (ITP) algoritme vartotojas gali savo nuožiūra rinktis tiek atsitiktinį sprendinių generavimą, tiek sprendinių formavimą panaudojant vieną iš konstrukcinių procedūrų.

Pradinis ITP algoritmo sprendinys generuojamas atsitiktiniu būdu. Mes remiamės prielaida, jog galutinių rezultatų gerumą lemia ne pradinis sprendinys, bet vėlesnis iteracinis sprendinių gerinimo procesas.

Sprendinių pagerinimas naudojant tabu paiešką nėra šio darbo pagrindinis tikslas, todėl šis ITP etapas čia detaliau nenagrinėjamas. Toliau aprašomos sprendinių mutavimo procedūros.

5.3.2 Sprendinių mutavimas

Sprendinių mutavimo paskirtis — diversifikuoti paieškos procesą, „išsklaidant“ paiešką kiek įmanoma įvairesnėse sprendinių erdvės dalyse. Mutavimo metu nėra tikslo duotojo sprendinio visiškai suardyti, sugriauti; greičiau atvirkščiai, generuojant naują sprendinį, siekiama, jog gautas sprendinys paveldėtų tam tikras geras ankstesnio sprendinio charakteristikas, bet tuo pačiu turi būti užtikrinamas ir pakankamas atstumas tarp esamo ir naujai gauto sprendinio.

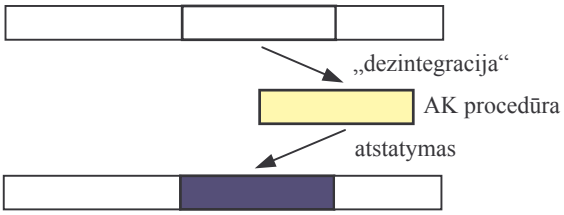
Sprendinių rekonstravimui siūloma naudoti „godžiojo“ pertvarkymo procedūrą. Šią procedūrą sudaro trys pagrindiniai žingsniai (žr. 7 pav.):

a) sprendinio „dezintegracija“ (dalinis suardymas);

b) sprendinio segmento (dalies) pertvarkymas panaudojant pasirinktą pagerinimo algoritmą;

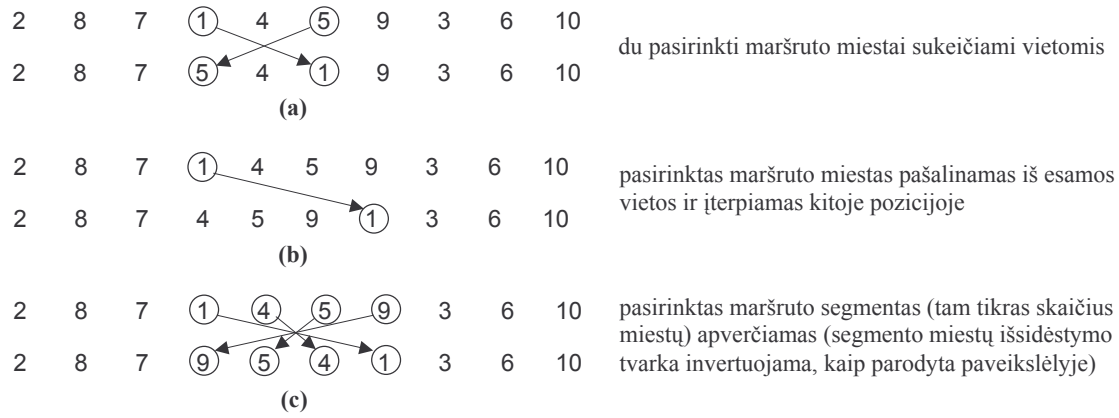
c) sprendinio atstatymas.

Pirmojo žingsnio metu sprendinys p savotiškai suardomas, išskaidomas. Tuomet gaunami sprendinio segmentai — atskiri daliniai sprendiniai. Gaunamas naujas, optimizuotas dalinis sprendinys p_r'. Trečiajame žingsnyje atliekamas sprendinio atstatymas, t.y. atskiri segmentai, tarp jų ir segmentas p_r', apjungiami į vieną sprendinį. Įvykdę šiuos tris žingsnius, turime rekonstruotą sprendinį p', kuris tarnaus kaip pradinis „taškas“ tolesniam procesui.



7 pav. Rekonstravimo procedūros grafinė interpretacija

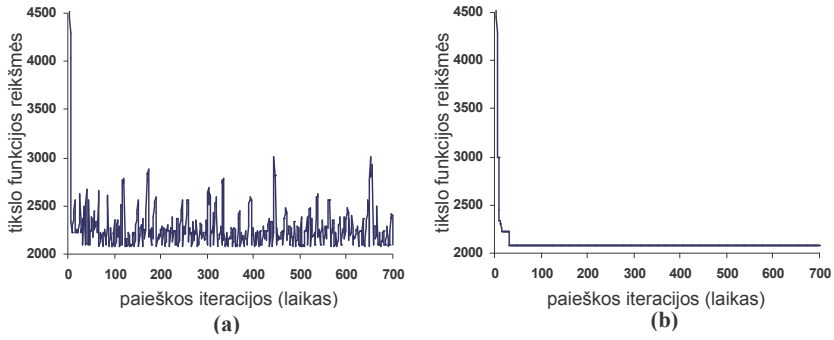
Sprendžiant komivojažieriaus uždavinį, įmanomi įvairūs sprendinių mutavimo būdai [7]. Galimų mutavimo operatorių pavyzdžiai yra: sukeitimo mutacija, įterpimo mutacija, inversijos (apvertimo) mutacija ir kt. Minėtų mutavimo operatorių iliustracijos pateiktos 8 pav.



8 pav. Mutavimo operatorių iliustravimas: sukeitimo mutacija (a), įterpimo mutacija (b), inversijos (apvertimo) mutacija (c)

Siekiant padidinti ITP algoritmo efektyvumo laipsnį, pasiūlyti patobulinti mutavimo operatoriai [8] — tai inversijos ir įterpimo mutavimo operatorius bei išplėstinis inversijos ir įterpimo mutavimo operatorius.

Kartu su aprašyta rekonstravimo procedūra gali būti taikomas ir alternatyvus paieškos diversifikavimas. Tokį papildomą diversifikavimą („šaltąjį“ paieškos restartą) tikslinga taikyti vadinamosios paieškos stagnacijos atvejais, t.y. tuomet, kai pakankamai ilgą laiko tarpą nepavyksta surasti geresnių (lokaliai optimalių) sprendinių (žr 9 pav.). Paprasčiausiu atveju alternatyvųjį rekonstravimą galime realizuoti tiesiog kaip naujo, visai atsitiktinio sprendinio generavimą.

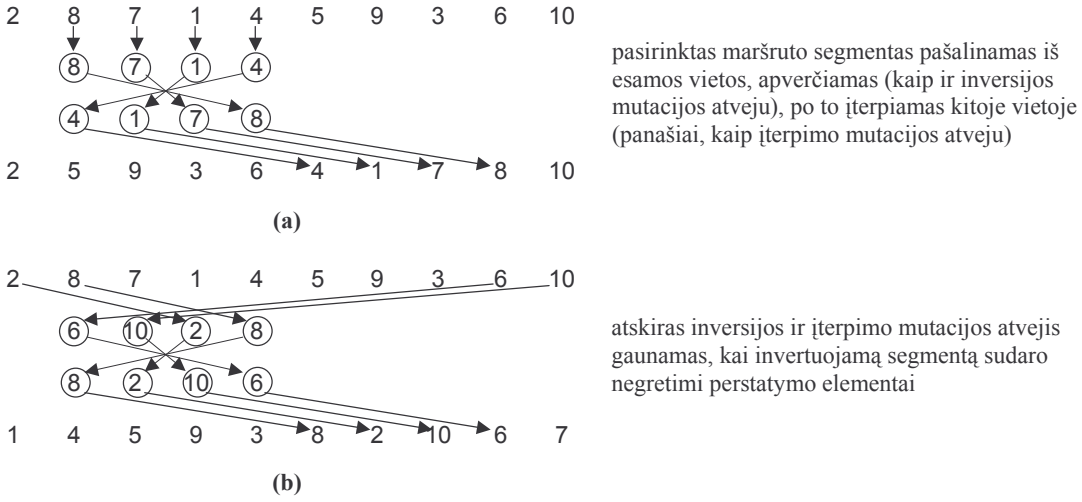


9 pav. Paieškos stagnacijos situacijos iliustracija:
(a) paieškos iteracijos ir kiekvienos iteracijos metu gautos tikslo funkcijos reikšmės; (b) paieškos iteracijos ir tikslo funkcijos reikšmės, atitinkančios geriausią tuo momentu rastą sprendinį

5.3.2.1 Inversijos ir įterpimo mutavimo operatorius

Inversijos ir įterpimo mutavimo (IĮM) operatoriuje [8] apjungtas sprendinio elementų invertavimas (apvertimas) ir invertuotų elementų įterpimas kitoje sprendinio pozicijoje. Be to, IĮM procedūroje numatytas papildomas inversijos ir įterpimo atvejis, kuris gaunamas, kai invertuojamą segmentą, t.y. sprendinio elementų (maršruto miestų) rinkinį sudaro negretimi sprendinio elementai – atsižvelgiama į tai, jog nagrinėjamas maršrutas būtų uždaras. Invertuojamų sprendinio elementų skaičius (segmento ilgis) (μ) apibrėžia mutavimo lygį (stiprumą). Tinkamo mutavimo lygio nustatymas yra labai svarbus, siekiant gauti aukštos kokybės sprendinius. Tai padaryti galima eksperimentų būdu (žr. 6.3 sk.).

IĮM operatoriaus veikimo principas iliustruojamas 10 pav. Formalus inversijos ir įterpimo mutavimo procedūros aprašymas pateiktas 11 pav.



10 pav. Inversijos ir įterpimo mutavimo operatoriaus iliustravimas:
standartinis inversijos atvejis (a), specialus inversijos atvejis (b)

```
procedure InversijosIterpimoMutavimas;
   // pradiniai duomenys: p – esamas sprendinys (maršrutas), n – miestų skaičius, μ – mutavimo lygis (stiprumas)//
 // rezultatai:  p~  – mutuotas sprendinys (maršrutas)//
begin
   tegul u, v (u ≠ v) – atsitiktiniai skaičiai iš intervalo [1, n],
     čia u yra pradinės invertuojamo segmento pozicijos maršrute p numeris, v –
įterpimo pozicijos numeris;
   if u + μ – 1 ≤ n   then kopijuoti(p, u, s, 1, μ)
   else begin kopijuoti(p, u, s, 1, n – u + 1);
              kopijuoti(p, 1, s, n – u + 2, μ – n + u + 1);
   end;
   s̄ := inversija(s) ;  // segmento s elementai invertuojami, gaunant segmentą s̄ //
   if v + μ – 1 ≤ n   then kopijuoti( s̄ , 1,  p~, v, μ)
   else begin kopijuoti( s̄ , 1,  p~, v, n – v + 1);
              kopijuoti( s̄ , n – v + 2, p~, 1, μ – n + v + 1)
   end;
   perkelti likusius elementus iš p į  p~
end.
```

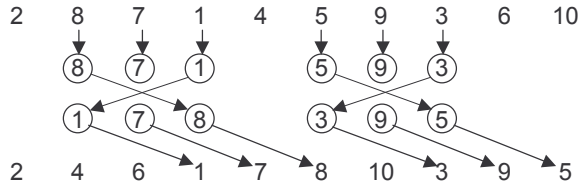
11 pav. Inversijos ir įterpimo mutavimo procedūros aprašymas.

Pastaba: procedūra kopijuoti(x, i, y, j, k) *perkelia k elementų iš maršruto* x*, pradedant i-taja pozicija, į maršrutą* y*, pradedant j-taja pozicija*

5.3.2.2 Išplėstinio inversijos ir įterpimo mutavimo operatorius

Bendresnis inversijos ir įterpimo mutavimo atvejis yra patobulinto, išplėstinio inversijos ir įterpimo mutavimo (IIĮM) operatorius [8]. Patobulinimo esmė — invertavimą ir įterpimą taikyti ne vienam, o keliems sprendinio, t.y. maršruto segmentams. Šiuo atveju yra svarbu deramo segmentų skaičiaus (N_s) parinkimas – priklauso nuo nagrinėjamo maršruto miestų skaičiaus dydžio: kuo didesnis miestų skaičius, tuo daugiau kartų bus atliekamas invertavimas ir įterpimas. IIĮM operatoriaus funkcionavimo schema iliustruojama 12 pav.

Parenkant sprendinį-kandidatą mutavimui, taikoma vadinamosios „koncentruotos paieškos“ strategija. Šiuo atveju kandidatas mutavimo procedūrai yra geriausias duotu momentu surastas lokaliai optimalus sprendinys. Tai leidžia fokusuoti, suintensyvinti paiešką tam tikrose sprendinių erdvės srityse, būtent elitinių lokaliai optimalių sprendinių aplinkose.



12 pav. Išplėstinio inversijos ir įterpimo mutavimo operatoriaus iliustravimas

5.4 Kitos iteratyviosios tabu paieškos algoritmo modifikacijos

Siekiant pagerinti sprendinius, buvo modifikuotas ir iteratyviosios tabu paieškos algoritmas:

1) Padidintas tabu paieškos atsitiktinumo (randomizavimo) lygis – kai atsitiktinumo lygio koeficientas lygus vienetui – vykdoma standartinė tabu paieška. Šį koeficientą sumažinus iki 0.95 – algoritme atsiranda daugiau atsitiktinių dydžių. Tačiau koeficientą sumažinus dar daugiau, algoritmo veikime atsiranda perdaug atsitiktinių dydžių, kurie gaunamus sprendinius nepagerina, bet pablogina.
2) Sumažinta tabu paieškos stagnacijos galimybė – įvedamas naujas tabu koeficientas, kurio dydis valdomas nustatymų byloje, bet algoritmo vykdymo metu jis yra susiejamas su vidinio ciklo iteracijų skaičiumi ir tiesiogiai įtakoja stagnacijos sąlygas – aspiracijos ir uždraudimų sąlygas.
3) Sumažinti uždraudimų (tabu) sąrašo atsitiktinumo dydžiai – algoritmas modifikuojamas taip, jog uždraudimo sąrašo dydis kinta palaipsniui pridedant po vieną vienetą, ir panaikinamas dydžio nustatymas pagal atsitiktinius skaičius.

Kiekvienai modifikacijai yra sudarytos atskiros algoritmo realizacijų versijos ir tęsiamos nuo paskutinės, kurios gauti rezultatai yra geresni negu prieš tai buvusių. Tokiu būdu algoritmo veikimo gerinimas vykdomas tolygiai. Be pačio algoritmų modifikacijų, dar buvo keičiami valdomosios bylos įvairūs nustatymai (koeficientai).

6. EKSPERIMENTAI IR JŲ REZULTATAI

Siekiant iš tirti sudarytų ITP modifikacijų efektyvumą, buvo atlikti eksperimentiniai tyrimai su įvairiais testiniais KU pavyzdžiais (duomenimis) iš KU testinių pavyzdžių elektroninės bibliotekos TSPLIB (angl. *Travelling Salesman Problem Library*) [26]. Konkrečiai buvo naudojami šie testiniai duomenys:

a280.dat	*eil101.dat*	*pr107.dat*
att48.dat	*gr17.dat*	*pr299.dat*
bays29.dat	*gr48.dat*	*rat99.dat*
berlin52.dat	*gr96.dat*	*rd100.dat*
bier127.dat	*kroa100.dat*	*si175.dat*
burma14.dat	*krob100.dat*	*st70.dat*
ch150.dat	*kroc100.dat*	*u159.dat*
d198.dat	*pr76.dat*	

Pastaba: skaičius pavadinime, pvz. 52,127 nurodo miestų skaičių.

Algoritmų įvertinimui naudojami žinomi optimalūs sprendiniai. Jų tikslo funkcijos optimalios reikšmės taip pat saugomos elektroninėje bibliotekoje TSPLIB.

Eksperimentų vykdymui programa (*OptiTSP – Optimizer for the TSP*) buvo parašyta Pascal kalba ir sukompiliuota FreePascal (versija 0.9.1) kompiliatoriumi. Programa vykdyta IBM tipo kompiuteriu, kuriame yra Intel firmos 3 GHz taktinio dažnio procesorius ir 512 MB operatyvinės atminties. Operacinė sistema – Windows XP Professional. Programos testavimo metu, kitos programos nebuvo aktyvios.

6.1 Eksperimentų vykdymas

Eksperimentai buvo vykdomi su atskiromis programos modifikacijomis (versijomis). Visoms versijoms buvo naudojami tokie parametrai: MS=10 (MS – multi startų skaičius), TIMELIMIT=0 (TIMELIMIT – laiko apribojimo požymis), RANDSEED=0123456789 (RANDSEED – atsitiktinių skaičių generavimas). Priklausomai nuo miestų skaičiaus ir duomenų bylos pobūdžio iteracijų skaičius parinktas toks, kad būtų gauti optimaliausi rezultatai ir užimtų mažiausiai laiko.

Paskutiniai trys eksperimentai buvo atlikti suvienodinus iteracijų skaičių visiems testiniams pavyzdžiams ir padvigubinus pakartotinių paleidimų (RS) („restartų”) skaičių.

6.2 Algoritmų efektyvumo vertinimo kriterijai

Pradinė algoritmo versija yra ITP_01, su ja lyginamos visos kitos versijos - nuo ITP_02 iki ITP_13. ITP_14- ITP_16 versijos naudojamos rezultatų palyginimui, kai visiems miestams suvienodinamas iteracijų skaičius ir padvigubinamas pakartotinių paleidimų skaičius.

Palyginimams buvo pasirinkti tokie efektyvumo kriterijai:

- vykdymo laikas *t.*
- vidutinis santykinis tikslo funkcijos (TF) nuokrypis nuo optimalios TF reikšmės — $\bar{\delta}$ ($\bar{\delta} = 100(\bar{z} - z_{\text{opt}})/z_{\text{opt}}$ [%], čia $\bar{z}$ yra gautų tikslo funkcijos reikšmių (maršrutų ilgių) vidurkis, apskaičiuotas atlikus 10 algoritmo pakartotinių vykdymų, o z_{opt} yra optimali tikslo funkcijos reikšmė (optimalias reikšmes galima sužinoti iš elektroninės bibliotekos TSPLIB));
- sprendinių, esančių „1% optimalumo intervale“ ($\bar{\delta} \le 1$), skaičius (kai atlikta 10 pakartotinių vykdymų) — $C_{1\%}$;
- visų paleidimų geriausio perstatymo tikslo funkcijos reikšmių, sutampančių (arba geresnių) su geriausia žinoma reikšme, kiekis C_{opt}.

Detalūs gauti rezultatai pateikti lentelėse 6.3 skyrelyje.

6.3 Eksperimentinių tyrimų rezultatai

Atliktų eksperimentinių tyrimų rezultatai pateikti 1-10 lentelėse ir 13-18 paveikslėliuose.

1 lentelė. Eksperimentinių tyrimų rezultatai

Testinio pvz. nr.	Testinio pavyzdžio pavad.	Miestų sk. (n)	z_{opt}	t, $\overline{\delta}$, $C_{1\%}/C_{opt}$								
				ITP_01 [1]			ITP_02 [2]			ITP_03 [3]		
1.	a280	280	2579	**4,5**	**2,06**	**10/1**	5,2	4,03	8/0	4,3	5,43	8/2
2.	att48	48	10628	0,1	1,05	10/8	**0**	**0,42**	**10/8**	**0**	**0,42**	**10/8**
3.	bays29	29	2020	**0**	**0**	**10/10**	**0**	**0**	**10/10**	**0**	**0**	**10/10**
4.	berlin52	52	7542	**0**	**0**	**10/10**	**0**	**0**	**10/10**	**0**	**0**	**10/10**
5.	bier127	127	118282	1,7	1,1	10/6	1,5	0,32	10/9	**1,3**	**0,12**	**10/9**
6.	burma14	14	3323	**0**	**0**	**10/10**	**0**	**0**	**10/10**	**0**	**0**	**10/10**
7.	ch150	150	6528	**1,3**	**1,11**	**10/6**	1,2	2,57	10/4	1	1,28	10/5
8.	d198	198	15780	**2,5**	**0,63**	**10/1**	2,7	1,76	10/1	2,4	0,78	10/2
9.	eil101	101	629	**1**	**1,59**	**10/5**	0,9	2,86	10/5	0,8	2,54	10/3
10.	gr17	17	2085	**0**	**0**	**10/10**	**0**	**0**	**10/10**	**0**	**0**	**10/10**
11.	gr48	48	5046	**0,2**	**0**	**10/10**	**0**	**0**	**10/10**	**0,1**	**0**	**10/10**
12.	gr96	96	55209	1,1	1,24	10/2	**0,8**	**0,89**	**10/4**	0,7	0,93	10/5
13.	kroa100	100	21282	0,2	0,47	10/8	0,1	1,15	10/6	**0,1**	**0**	**10/10**
14.	krob100	100	22141	0,5	1,65	10/4	0,4	1,91	10/4	**0,3**	**1,39**	**10/6**
15.	kroc100	100	20749	**0,4**	**0**	**10/10**	0,5	0,5	10/9	0,4	0,61	10/8
16.	pr76	76	108159	0,2	0,97	10/8	**0,1**	**0**	**10/10**	**0,2**	**0**	**10/10**
17.	pr107	107	44303	**0,9**	**0,87**	**10/6**	0,9	1,37	10/4	0,8	1,29	10/6
18.	pr299	299	48191	9,5	2,15	10/0	11,1	2,66	10/0	**9**	**1,83**	**10/2**
19.	rat99	99	1211	0,9	1,4	10/6	**0,9**	**0,33**	**10/7**	0,6	0,41	10/6
20.	rd100	100	7910	0,7	0,86	10/8	**0,5**	**0**	**10/10**	0,5	0,43	10/9
21.	si175	175	21407	1,7	0,4	10/2	**3,1**	**0,36**	**10/0**	1,5	0,4	10/3
22.	st70	70	675	**0,7**	**0**	**10/10**	**0,4**	**0**	**10/10**	**0,3**	**0**	**10/10**
23.	u159	159	42080	1,5	0,75	10/9	**1,5**	**0**	**10/10**	**1,2**	**0**	**10/10**
Vidurkis				1,2870	0,7957		1,3826	0,9187		**1,1087**	**0,7765**	

[1] ITP_01 – pradinis iteratyviosios tabu paieškos variantas

[2] ITP_02 – ITP su įterpimo mutacija, kai segmentų sk. lygus 1

[3] ITP_03 – ITP su inversijos ir įterpimo mutacija, kai segmentų sk. lygus 3

2 lentelė. Eksperimentinių tyrimų rezultatai

Testinio pvz. nr.	Testinio pavyzdžio pavad	Miestų sk. (n)	z_{opt}	t, $\overline{\delta}$, $C_{1\%}/C_{opt}$					
				ITP_04[4]			ITP_05[5]		
1.	a280	280	2579	**4,4**	**2,4**	**9/3**	4,8	4,07	9/3
2.	att48	48	10628	**0,1**	**0,42**	**10/8**	**0,1**	**0,42**	**10/8**
3.	bays29	29	2020	**0**	**0**	**10/10**	**0**	**0**	**10/10**
4.	berlin52	52	7542	**0,1**	**0**	**10/10**	**0,1**	**0**	**10/10**
5.	bier127	127	118282	**1,3**	**0,3**	**10/8**	1,5	0,32	10/9
6.	burma14	14	3323	**0**	**0**	**10/10**	**0**	**0**	**10/10**
7.	ch150	150	6528	1	0,46	10/8	**1**	**0,29**	**10/9**
8.	d198	198	15780	**2,4**	**1,13**	**10/0**	3	1,5	10/1
9.	eil101	101	629	0,8	3,18	9/5	**0,9**	**2,07**	**10/6**
10.	gr17	17	2085	**0**	**0**	**10/10**	**0,1**	**0**	**10/10**
11.	gr48	48	5046	**0,1**	**0**	**10/10**	**0,2**	**0**	**10/10**
12.	gr96	96	55209	0,7	1,45	10/4	**1**	**1,24**	**10/3**
13.	kroa100	100	21282	0,1	0,68	10/7	**0,1**	**0,32**	**10/7**
14.	krob100	100	22141	0,3	1,3	10/6	**0,3**	**0,52**	**10/8**
15.	kroc100	100	20749	0,3	1,1	10/7	**0,4**	**0**	**10/10**
16.	pr76	76	108159	**0,2**	**0**	**10/10**	**0,1**	**0**	**10/10**
17.	pr107	107	44303	0,8	0,99	10/7	**1**	**0,3**	**10/9**
18.	pr299	299	48191	9	2,49	10/0	**11,4**	**1,89**	**10/3**
19.	rat99	99	1211	**0,6**	**0,25**	**10/8**	0,6	1,98	10/5
20.	rd100	100	7910	**0,5**	**0**	**10/10**	**0,5**	**0**	**10/10**
21.	si175	175	21407	**1,5**	**0,25**	**10/5**	1,8	0,36	10/0
22.	st70	70	675	**0,4**	**0**	**10/10**	**0,6**	**0**	**10/10**
23.	u159	159	42080	**1,2**	**0**	**10/10**	1,5	0,75	10/9
			Vidurkis	**1,1217**	0,7130		1,3478	**0,6970**	

[4] ITP_04 – ITP su inversijos ir įterpimo mutacija, kai segmentų sk. lygus 5

[5] ITP_05 – ITP su išplėstinė inversijos ir įterpimo mutacija, kai segmentų sk. lygus 5

3 lentelė. Eksperimentinių tyrimų rezultatai

Testinio pvz. nr.	Testinio pavyzdžio pavad	Miestų sk. (n)	z_{opt}	t, $\overline{\delta}$, $C_{1\%}/C_{opt}$					
				ITP_06[6]			ITP_07[7]		
1.	a280	280	2579	4,6	6,32	9/0	**4,6**	**3,14**	**9/3**
2.	att48	48	10628	**0,1**	**0,42**	**10/8**	**0,1**	**0,42**	**10/8**
3.	bays29	29	2020	**0,1**	**0**	**10/10**	**0**	**0**	**10/10**
4.	berlin52	52	7542	**0**	**0**	**10/10**	**0,1**	**0**	**10/10**
5.	bier127	127	118282	**1,5**	**0,41**	**10/9**	1,5	0,76	10/7
6.	burma14	14	3323	**0**	**0**	**10/10**	**0**	**0**	**10/10**
7.	ch150	150	6528	1,2	1,57	10/6	**1,1**	**0,98**	**10/6**
8.	d198	198	15780	2,6	1,16	10/0	**2,7**	**0,87**	**10/2**
9.	eil101	101	629	**0,9**	**1,75**	**10/4**	0,9	2,23	10/5
10.	gr17	17	2085	**0**	**0**	**10/10**	**0,1**	**0**	**10/10**
11.	gr48	48	5046	**0,1**	**0**	**10/10**	**0,1**	**0**	**10/10**
12.	gr96	96	55209	0,9	1,45	10/4	0,9	1,41	10/3
13.	kroa100	100	21282	**0,1**	**0,86**	**10/7**	0,1	1,12	10/5
14.	krob100	100	22141	0,4	1,14	10/6	**0,3**	**0,52**	**10/8**
15.	kroc100	100	20749	0,4	0,99	10/8	**0,4**	**0,1**	**10/9**
16.	pr76	76	108159	**0,2**	**0**	**10/10**	**0,2**	**0**	**10/10**
17.	pr107	107	44303	1	1,52	10/5	**1**	**1,48**	**10/6**
18.	pr299	299	48191	10	2,64	10/0	**10**	**1,93**	**10/1**
19.	rat99	99	1211	0,6	0,17	10/9	**0,6**	**0,08**	**10/9**
20.	rd100	100	7910	**0,5**	**0**	**10/10**	**0,5**	**0**	**10/10**
21.	si175	175	21407	**1,8**	**0,21**	**10/7**	1,8	0,35	10/4
22.	st70	70	675	0,6	0,74	10/9	**0,6**	**0**	**10/10**
23.	u159	159	42080	**1,5**	**0**	**10/10**	**1,5**	**0**	**10/10**
			Vidurkis	1,2652	0,9283		1,2652	**0,6691**	

[6] ITP_06 – ITP su inversijos mutacija, kai segmentų sk. lygus 5

[7] ITP_07 – ITP su įterpimo mutacija, kai segmentų sk. lygus 5

4 lentelė. Eksperimentinių tyrimų rezultatai

Testinio pvz. nr.	Testinio pavyzdžio pavad.	Miestų sk. (n)	z_{opt}	Inversijos ir įterpimo mutacija, kai segmentų sk. lygus 5								
				t, $\overline{\delta}$, $C_{1\%}/C_{opt}$								
				ITP_08[8]			ITP_04[9]			ITP_09[10]		
1.	a280	280	2579	4,6	3,88	8/3	**4,4**	**2,4**	**9/3**	4,3	6,9	7/1
2.	att48	48	10628	0,1	0,56	10/7	0,1	0,42	10/8	**0,1**	**0,19**	**10/9**
3.	bays29	29	2020	**0**	**0**	**10/10**	**0**	**0**	**10/10**	**0,1**	**0**	**10/10**
4.	berlin52	52	7542	**0,1**	**0**	**10/10**	**0,1**	**0**	**10/10**	**0**	**0**	**10/10**
5.	bier127	127	118282	1,5	0,32	10/9	1,3	0,3	10/8	**1,3**	**0,15**	**10/9**
6.	burma14	14	3323	**0,1**	**0**	**10/10**	**0**	**0**	**10/10**	**0,1**	**0**	**10/10**
7.	ch150	150	6528	1,1	1,51	10/4	**1**	**0,46**	**10/8**	1	0,98	10/6
8.	d198	198	15780	2,6	0,94	10/1	2,4	1,13	10/0	**2,3**	**0,87**	**10/1**
9.	eil101	101	629	0,9	3,82	9/5	0,8	3,18	9/5	**0,8**	**3,02**	**8/6**
10.	gr17	17	2085	**0**	**0**	**10/10**	**0**	**0**	**10/10**	**0**	**0**	**10/10**
11.	gr48	48	5046	**0,1**	**0**	**10/10**	**0,1**	**0**	**10/10**	**0,2**	**0**	**10/10**
12.	gr96	96	55209	0,9	1,45	10/4	0,7	1,45	10/4	**1**	**0,74**	**10/5**
13.	kroa100	100	21282	0,2	1,13	10/5	**0,1**	**0,68**	**10/7**	0,1	0,88	10/7
14.	krob100	100	22141	0,4	0,94	10/7	0,3	1,3	10/6	**0,3**	**0,7**	**10/8**
15.	kroc100	100	20749	**0,4**	**0,3**	**10/9**	0,3	1,1	10/7	**0,4**	**0,3**	**10/9**
16.	pr76	76	108159	**0,2**	**0**	**10/10**	**0,2**	**0**	**10/10**	**0,2**	**0**	**10/10**
17.	pr107	107	44303	**0,9**	**0,91**	**10/7**	0,8	0,99	10/7	0,9	1,48	10/5
18.	pr299	299	48191	10,2	2,57	10/0	**9**	**2,49**	**10/0**	9,6	2,62	10/0
19.	rat99	99	1211	**0,6**	**0**	**10/10**	0,6	0,25	10/8	0,6	0,33	10/7
20.	rd100	100	7910	0,5	0,43	10/9	**0,5**	**0**	**10/10**	0,5	0,43	10/9
21.	si175	175	21407	**1,7**	**0,23**	**10/5**	1,5	0,25	10/5	1,8	0,33	10/3
22.	st70	70	675	**0,6**	**0**	**10/10**	**0,4**	**0**	**10/10**	**0,6**	**0**	**10/10**
23.	u159	159	42080	**1,5**	**0**	**10/10**	**1,2**	**0**	**10/10**	1,4	0,75	10/9
			Vidurkis	1,2696	0,8257		1,1217	**0,7130**		**1,2000**	0,8987	

[8] ITP_08 – ITP, kai atsitiktinių dydžių lygis lygus 1,1

[9] ITP_04 – ITP, kai atsitiktinių dydžių lygis lygus 0,95

[10] ITP_09 – ITP, kai atsitiktinių dydžių lygis lygus 0,85

5 lentelė. Eksperimentinių tyrimų rezultatai

Testinio pvz. nr.	Testinio pavyz-džio pavad.	Miestų sk. (n)	z_{opt}	Inversijos ir įterpimo mutacija					
				t, $\overline{\delta}$, $C_{1\%}/C_{opt}$					
				ITP_10_1[11]			ITP_11_1[12]		
1.	a280	280	2579	**4,4**	**2,44**	**9/1**	4,9	4,54	8/1
2.	att48	48	10628	**0,1**	**0,42**	**10/8**	**0,1**	**0,42**	**10/8**
3.	bays29	29	2020	**0**	**0**	**10/10**	**0**	**0**	**10/10**
4.	berlin52	52	7542	**0,1**	**0**	**10/10**	**0**	**0**	**10/10**
5.	bier127	127	118282	1,3	0,3	10/8	**1,5**	**0,12**	**10/9**
6.	burma14	14	3323	**0,1**	**0**	**10/10**	**0**	**0**	**10/10**
7.	ch150	150	6528	**1,1**	**0,46**	**10/8**	1,3	1,28	10/5
8.	d198	198	15780	2,4	0,92	10/0	**2,5**	**0,7**	**10/1**
9.	eil101	101	629	0,8	3,18	9/5	**0,8**	**2,54**	**10/3**
10.	gr17	17	2085	**0**	**0**	**10/10**	**0**	**0**	**10/10**
11.	gr48	48	5046	**0,1**	**0**	**10/10**	**0,1**	**0**	**10/10**
12.	gr96	96	55209	1	1,45	10/4	**0,9**	**0,93**	**10/5**
13.	kroa100	100	21282	0,1	0,68	10/7	**0,1**	**0**	**10/10**
14.	krob100	100	22141	**0,4**	**1,3**	**10/6**	0,4	1,39	10/6
15.	kroc100	100	20749	0,3	1,1	10/7	**0,3**	**0,61**	**10/8**
16.	pr76	76	108159	**0,1**	**0**	**10/10**	0,2	0	10/10
17.	pr107	107	44303	**1**	**0,99**	**10/7**	0,9	1,29	10/6
18.	pr299	299	48191	10,2	2,74	10/0	**9,4**	**1,44**	**10/1**
19.	rat99	99	1211	**0,6**	**0,25**	**10/8**	0,5	0,41	10/6
20.	rd100	100	7910	**0,6**	**0**	**10/10**	0,6	0,43	10/9
21.	si175	175	21407	**1,8**	**0,27**	**10/3**	1,5	0,42	10/3
22.	st70	70	675	**0,5**	**0**	**10/10**	**0,5**	**0**	**10/10**
23.	u159	159	42080	1,5	0,18	10/9	**1,3**	**0**	**10/10**
			Vidurkis	1,2391	0,7252		**1,2087**	**0,7183**	

[11] ITP_10_1 – ITP, kai papildomas tabu koeficientas (Pap_koef_tabu) lygus 0.1, o segmentų sk. lygus 5

[12] ITP_11_1 – ITP, kai papildomas tabu koeficientas (Pap_koef_tabu) lygus 0.1, o segmentų sk. lygus 3

6 lentelė. Eksperimentinių tyrimų rezultatai

Testinio pvz. nr.	Testinio pavyz-džio pavad	Miestų sk. (n)	z_{opt}	Inversijos ir įterpimo mutacija					
				t, $\overline{\delta}$, $C_{1\%}/C_{opt}$					
				ITP_10_2[13]			ITP_11_2[14]		
1.	a280	280	2579	**4,7**	**4,38**	**8/1**	5	7,37	7/1
2.	att48	48	10628	**0,1**	**0,42**	**10/8**	**0,1**	**0,42**	**10/8**
3.	bays29	29	2020	**0**	**0**	**10/10**	**0**	**0**	**10/10**
4.	berlin52	52	7542	**0**	**0**	**10/10**	**0**	**0**	**10/10**
5.	bier127	127	118282	1,6	0,59	10/7	**1,4**	**0**	**10/10**
6.	burma14	14	3323	**0,1**	**0**	**10/10**	**0**	**0**	**10/10**
7.	ch150	150	6528	**1,2**	**0,46**	**10/8**	1,2	2,41	10/2
8.	d198	198	15780	**2,6**	**0,58**	**10/1**	2,4	0,77	10/0
9.	eil101	101	629	0,9	2,86	9/6	**1**	**2,38**	**9/5**
10.	gr17	17	2085	**0**	**0**	**10/10**	**0,1**	**0**	**10/10**
11.	gr48	48	5046	**0,1**	**0**	**10/10**	**0,1**	**0**	**10/10**
12.	gr96	96	55209	0,9	1,64	10/4	**0,9**	**1,23**	**10/3**
13.	kroa100	100	21282	0,4	0,74	10/7	**0,1**	**0,05**	**10/9**
14.	krob100	100	22141	0,4	1,58	10/5	**0,4**	**1,13**	**10/7**
15.	kroc100	100	20749	0,4	1,3	10/7	**0,4**	**0,3**	**10/9**
16.	pr76	76	108159	**0,2**	**0**	**10/10**	**0,2**	**0**	**10/10**
17.	pr107	107	44303	0,9	1,1	10/6	**0,9**	**0,61**	**10/8**
18.	pr299	299	48191	9,8	2,53	10/0	**9,6**	**1,23**	**10/2**
19.	rat99	99	1211	**0,6**	**0,33**	**10/7**	0,6	0,58	10/5
20.	rd100	100	7910	**0,6**	**0**	**10/10**	0,5	0,43	10/9
21.	si175	175	21407	**1,8**	**0,22**	**10/5**	1,5	0,42	10/3
22.	st70	70	675	**0,6**	**0**	**10/10**	**0,5**	**0**	**10/10**
23.	u159	159	42080	**1,5**	**0**	**10/10**	0,8	1,5	10/8
			Vidurkis	1,2783	**0,8143**		**1,2043**	0,9057	

[13] ITP_10_2 – ITP, kai papildomas tabu koeficientas (Pap_koef_tabu) lygus 0.25, o segmentų sk.lygus 5

[14] ITP_11_2 – ITP, kai papildomas tabu koeficientas (Pap_koef_tabu) lygus 0.25, o segmentų sk.lygus 3

7 lentelė. Eksperimentinių tyrimų rezultatai

Testinio pvz. nr.	Testinio pavyz-džio pavad	Miestų sk. (n)	z_{opt}	Inversijos ir įterpimo mutacija					
				t, $\overline{\delta}$, $C_{1\%}/C_{opt}$					
				ITP_10_3[15]			ITP_11_3[16]		
1.	a280	280	2579	**4,5**	**4,38**	**8/1**	4,9	7,72	7/1
2.	att48	48	10628	**0,1**	**0,42**	**10/8**	**0,1**	**0,42**	**10/8**
3.	bays29	29	2020	**0,1**	**0**	**10/10**	**0,1**	**0**	**10/10**
4.	berlin52	52	7542	**0**	**0**	**10/10**	**0,1**	**0**	**10/10**
5.	bier127	127	118282	1,5	0,59	10/7	**1,4**	**0,12**	**10/9**
6.	burma14	14	3323	**0**	**0**	**10/10**	**0,1**	**0**	**10/10**
7.	ch150	150	6528	**1,1**	**0,46**	**10/8**	1	2,41	10/2
8.	d198	198	15780	**2,7**	**0,58**	**10/1**	2,6	0,86	10/0
9.	eil101	101	629	0,9	2,86	9/6	**0,9**	**2,38**	**9/5**
10.	gr17	17	2085	**0,1**	**0**	**10/10**	**0**	**0**	**10/10**
11.	gr48	48	5046	**0,1**	**0**	**10/10**	**0,1**	**0**	**10/10**
12.	gr96	96	55209	0,9	1,64	10/4	**0,8**	**1,23**	**10/3**
13.	kroa100	100	21282	0,2	0,74	10/7	**0,2**	**0,05**	**10/9**
14.	krob100	100	22141	0,3	1,58	10/5	**0,4**	**1,13**	**10/7**
15.	kroc100	100	20749	0,3	1,3	10/7	**0,4**	**0,3**	**10/9**
16.	pr76	76	108159	**0,1**	**0**	**10/10**	**0,1**	**0**	**10/10**
17.	pr107	107	44303	1	1,1	10/6	**0,9**	**0,61**	**10/8**
18.	pr299	299	48191	9,9	2,53	10/0	**9,6**	**1,23**	**10/2**
19.	rat99	99	1211	**0,6**	**0,33**	**10/7**	0,5	0,58	10/5
20.	rd100	100	7910	**0,6**	**0**	**10/10**	0,5	0,43	10/9
21.	si175	175	21407	**1,5**	**0,22**	**10/5**	1,7	0,45	10/3
22.	st70	70	675	**0,5**	**0**	**10/10**	**0,5**	**0**	**10/10**
23.	u159	159	42080	**1,2**	**0**	**10/10**	1,5	1,5	10/8
			Vidurkis	**1,2261**	**0,8143**		1,2348	0,9313	

[15] ITP_10_3 – ITP, kai papildomas tabu koeficientas (Pap_koef_tabu) lygus 0.5, o segmentų sk.lygus 5

[16] ITP_11_3 – ITP, kai papildomas tabu koeficientas (Pap_koef_tabu) lygus 0.5, o segmentų sk.lygus 3

8 lentelė. Eksperimentinių tyrimų rezultatai

Testinio pvz. nr.	Testinio pavyzdžio pavad	Miestų sk. (n)	z_{opt}	Inversijos ir įterpimo mutacija					
				t, $\overline{\delta}$, $C_{1\%}/C_{opt}$					
				ITP_10_4[17]			ITP_11_4[18]		
1.	a280	280	2579	**4,9**	**2,6**	**9/1**	4,1	4,46	8/1
2.	att48	48	10628	**0,1**	**0,42**	**10/8**	**0,1**	**0,42**	**10/8**
3.	bays29	29	2020	**0**	**0**	**10/10**	**0,1**	**0**	**10/10**
4.	berlin52	52	7542	**0**	**0**	**10/10**	**0**	**0**	**10/10**
5.	bier127	127	118282	1,6	0,3	10/8	**1,3**	**0,12**	**10/9**
6.	burma14	14	3323	**0**	**0**	**10/10**	**0**	**0**	**10/10**
7.	ch150	150	6528	**1,2**	**0,46**	**10/8**	1	1,28	10/5
8.	d198	198	15780	2,7	0,87	10/0	**2,3**	**0,7**	**10/1**
9.	eil101	101	629	0,8	3,18	9/5	**0,8**	**2,54**	**10/3**
10.	gr17	17	2085	**0**	**0**	**10/10**	**0,1**	**0**	**10/10**
11.	gr48	48	5046	**0,1**	**0**	**10/10**	**0,1**	**0**	**10/10**
12.	gr96	96	55209	0,9	1,45	10/4	**0,8**	**0,93**	**10/5**
13.	kroa100	100	21282	0,2	0,68	10/7	**0,1**	**0**	**10/10**
14.	krob100	100	22141	**0,4**	**1,3**	**10/6**	0,3	1,39	10/6
15.	kroc100	100	20749	0,4	1,1	10/7	**0,4**	**0,61**	**10/8**
16.	pr76	76	108159	**0,2**	**0**	**10/10**	**0,2**	**0**	**10/10**
17.	pr107	107	44303	**1**	**0,99**	**10/7**	0,8	1,29	10/6
18.	pr299	299	48191	10,5	2,49	10/0	**8,6**	**1,44**	**10/1**
19.	rat99	99	1211	**0,7**	**0,25**	**10/8**	0,6	0,41	10/6
20.	rd100	100	7910	**0,6**	**0**	**10/10**	**0,5**	**0,43**	**10/9**
21.	si175	175	21407	**1,8**	**0,27**	**10/3**	1,5	0,42	10/3
22.	st70	70	675	**0,5**	**0**	**10/10**	**0,5**	**0**	**10/10**
23.	u159	159	42080	1,5	0,18	10/9	**1,3**	**0**	**10/10**
			Vidurkis	1,3087	0,7191		**1,1087**	**0,7148**	

[17]ITP_10_4 - ITP, kai papildomas tabu koeficientas (Pap_koef_tabu) lygus 0.095, o segmentų sk.lygus 5

[18]ITP_11_4 - ITP, kai papildomas tabu koeficientas (Pap_koef_tabu) lygus 0.095, o segmentų sk.lygus 3

9 lentelė. Eksperimentinių tyrimų rezultatai

Testinio pvz. nr.	Testinio pavyzdžio pavad	Miestų sk. (n)	z_{opt}	Inversijos ir įterpimo mutacija								
				t, $\overline{\delta}$, $C_{1\%}/C_{opt}$								
				ITP_12[19]			ITP_13[20]			ITP_03[21]		
1.	a280	280	2579	4,9	4,27	8/1	**4,6**	**2,17**	**10/2**	4,3	5,43	8/2
2.	att48	48	10628	0,1	0,47	10/8	**0,1**	**0,42**	**10/8**	**0**	**0,42**	**10/8**
3.	bays29	29	2020	**0**	**0**	**10/10**	**0**	**0**	**10/10**	**0**	**0**	**10/10**
4.	berlin52	52	7542	**0**	**0**	**10/10**	**0**	**0**	**10/10**	**0**	**0**	**10/10**
5.	bier127	127	118282	**1,4**	**0,04**	**10/9**	**1,3**	**0,04**	**10/9**	1,3	0,12	10/9
6.	burma14	14	3323	**0,1**	**0**	**10/10**	**0**	**0**	**10/10**	**0**	**0**	**10/10**
7.	ch150	150	6528	1,1	1,15	10/5	**1**	**0,52**	**10/8**	1	1,28	10/5
8.	d198	198	15780	2,5	0,99	10/0	2,4	1,02	10/1	**2,4**	**0,78**	**10/2**
9.	eil101	101	629	0,9	2,7	9/6	**0,8**	**2,54**	**9/6**	**0,8**	**2,54**	**10/3**
10.	gr17	17	2085	**0**	**0**	**10/10**	**0**	**0**	**10/10**	**0**	**0**	**10/10**
11.	gr48	48	5046	0,1	0,18	10/9	**0,1**	**0**	**10/10**	**0,1**	**0**	**10/10**
12.	gr96	96	55209	1	1,08	10/4	0,7	1,08	10/4	**0,7**	**0,93**	**10/5**
13.	kroa100	100	21282	**0,2**	**0**	**10/10**	**0,1**	**0**	**10/10**	**0,1**	**0**	**10/10**
14.	krob100	100	22141	**0,3**	**1,2**	**10/6**	0,4	1,83	9/6	0,3	1,39	10/6
15.	kroc100	100	20749	0,4	0,5	10/9	**0,3**	**0**	**10/10**	0,4	0,61	10/8
16.	pr76	76	108159	**0,1**	**0**	**10/10**	**0,2**	**0**	**10/10**	**0,2**	**0**	**10/10**
17.	pr107	107	44303	1	1,58	10/5	**0,8**	**0,99**	**10/6**	0,8	1,29	10/6
18.	pr299	299	48191	9,9	2,96	10/0	11,5	2,24	10/0	**9**	**1,83**	**10/2**
19.	rat99	99	1211	**0,6**	**0,33**	**10/7**	0,5	0,41	10/6	0,6	0,41	10/6
20.	rd100	100	7910	**0,6**	**0**	**10/10**	0,5	0,86	10/8	0,5	0,43	10/9
21.	si175	175	21407	**1,7**	**0,28**	**10/3**	1,6	0,31	10/4	1,5	0,4	10/3
22.	st70	70	675	0,6	0,89	10/9	**0,4**	**0**	**10/10**	**0,3**	**0**	**10/10**
23.	u159	159	42080	**1,3**	**0**	**10/10**	**1,4**	**0**	**10/10**	**1,2**	**0**	**10/10**
Vidurkis				1,2522	0,8096		1,2478	**0,6274**		**1,1087**	0,7765	

[19] ITP_12 – ITP, kai viršutinė ir apatinė uždraudimų sąrašo ribos (LTLS_FACT ir HTLS_FACT) yra lygios 0.090. Segmentų sk. lygus 3

[20] ITP_13 – ITP, kai viršutinė (LTLS_FACT) uždraudimų sąrašo riba lygi 0,090, o apatinė (HTLS_FACT) uždraudimų sąrašo riba lygi 0.130. Segmentų sk. lygus 3

[21] ITP_03 – ITP, kai viršutinė (LTLS_FACT) uždraudimų sąrašo riba lygi 0,110, o apatinė (HTLS_FACT) uždraudimų sąrašo riba lygi 0.130. Segmentų sk. lygus 3

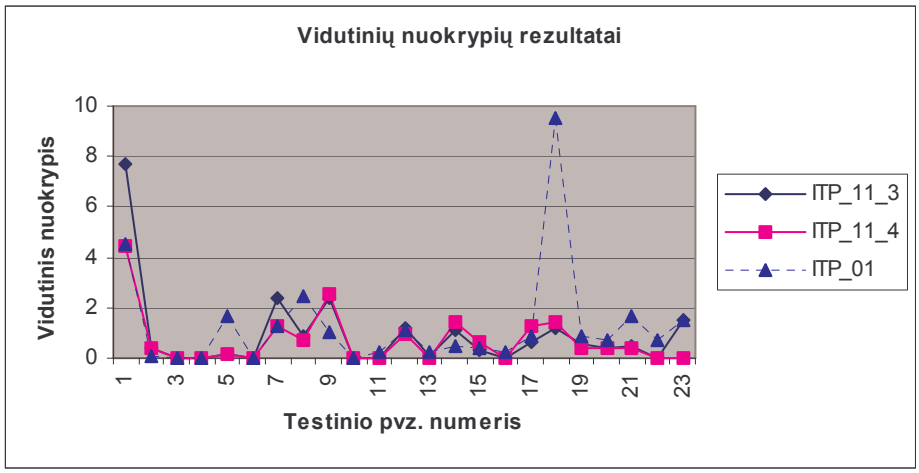
10 lentelė. Eksperimentinių tyrimų rezultatai

Testinio pvz. nr.	Testinio pavyzdžio pavad	Miestų sk. (n)	z_{opt}	Inversijos ir įterpimo mutacija								
				t, $\overline{\delta}$, $C_{1\%}/C_{opt}$								
				ITP_14[22]			ITP_15[23]			ITP_16[24]		
1.	a280	280	2579	3,2	3,92	10/1	**8,2**	**3,1**	**10/1**	5,5	5,27	9/0
2.	att48	48	10628	0,1	0,42	10/8	**0,2**	**0,19**	**10/9**	0,8	0,38	10/8
3.	bays29	29	2020	**0**	**0**	**10/10**	**0,1**	**0**	**10/10**	**0,1**	**0**	**10/10**
4.	berlin52	52	7542	**0,1**	**0**	**10/10**	**0,3**	**0**	**10/10**	**0,1**	**0**	**10/10**
5.	bier127	127	118282	0,5	1,39	10/4	**1,3**	**0,7**	**10/4**	0,8	1,49	10/2
6.	burma14	14	3323	**0,1**	**0**	**10/10**	**0**	**0**	**10/10**	**0**	**0**	**10/10**
7.	ch150	150	6528	0,5	3,32	9/4	**1,4**	**1,51**	**10/4**	0,8	2,93	10/1
8.	d198	198	15780	1,4	1,93	10/1	**4,2**	**0,58**	**10/1**	2,2	1,41	10/0
9.	eil101	101	629	0,2	3,97	9/1	**0,7**	**1,91**	**10/4**	0,4	3,82	9/3
10.	gr17	17	2085	**0**	**0**	**10/10**	**0,1**	**0**	**10/10**	**0,1**	**0**	**10/10**
11.	gr48	48	5046	0,1	0,36	10/8	**0,2**	**0**	**10/10**	0,1	0,18	10/9
12.	gr96	96	55209	0,3	1,74	10/4	**0,7**	**1,35**	**10/3**	0,3	1,39	10/3
13.	kroa100	100	21282	0,2	0,11	10/9	**0,7**	**0**	**10/10**	0,4	0,2	10/8
14.	krob100	100	22141	0,3	2,19	9/5	**0,7**	**0**	**10/10**	0,4	0,17	10/9
15.	kroc100	100	20749	0,3	0,02	10/9	**0,8**	**0**	**10/10**	0,4	0,52	10/6
16.	pr76	76	108159	**0,1**	**0**	**10/10**	**0,4**	**0**	**10/10**	**0,2**	**0**	**10/10**
17.	pr107	107	44303	0,2	1,19	10/6	**0,8**	**0,85**	**10/6**	0,4	2,11	10/2
18.	pr299	299	48191	4,2	5,08	8/0	**13,6**	**3,06**	**10/0**	7	6,15	9/0
19.	rat99	99	1211	0,3	1,57	10/2	**0,7**	**0,5**	**10/6**	0,4	1,49	10/4
20.	rd100	100	7910	0,2	2,14	10/5	**0,7**	**0,43**	**10/9**	0,4	0,51	10/8
21.	si175	175	21407	0,8	0,52	10/1	**2,8**	**0,39**	**10/2**	1,3	0,51	10/1
22.	st70	70	675	0,2	1,48	10/8	**0,3**	**0**	**10/10**	0,2	0,74	10/9
23.	u159	159	42080	0,8	1,54	10/8	**2,4**	**0**	**10/10**	**1,2**	**0**	**10/10**
Vidurkis				**0,6130**	1,4300		1,7957	**0,6335**		1,0217	1,2726	

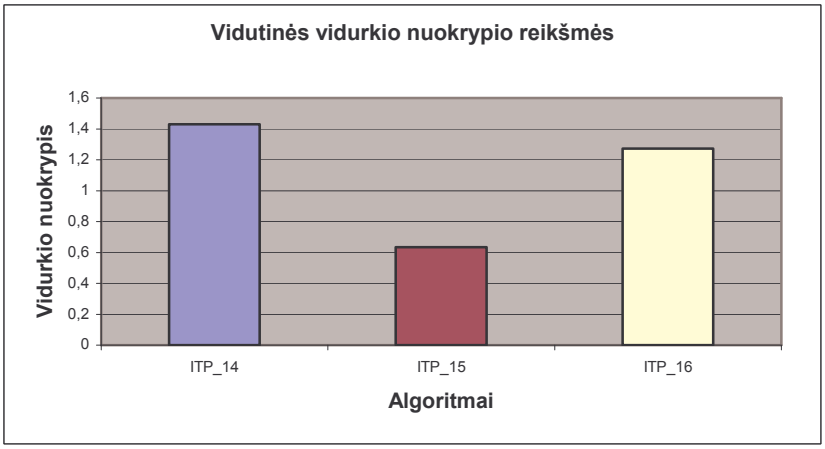
[22] ITP_14 – ITP, kai visiems testiniams pavyzdžiams suvienodinamas iteracijų skaičius (ITER_1=100). Segmentų sk. lygus 3

[23] ITP_15 – ITP, kai visiems testiniams pavyzdžiams suvienodinamas iteracijų skaičius (ITER_1=100) ir padvigubinamas pakartotinis vykdimų skaičius (RS=2). Segmentų sk. lygus 3

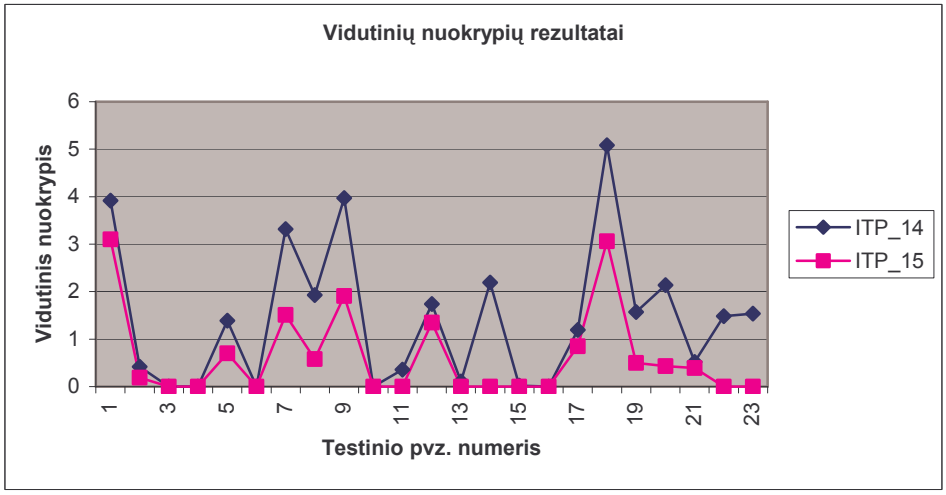
[24] ITP_16 – ITP, kai visiems testiniams pavyzdžiams suvienodinamas iteracijų skaičius (ITER_1=50) ir padvigubinamas pakartotinis vykdimų skaičius (RS=2). Segmentų sk. lygus 3



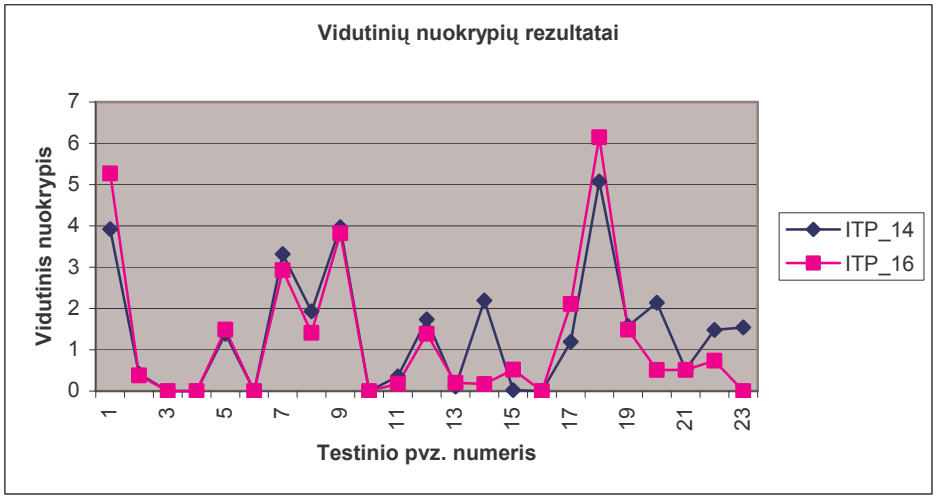
13 pav. Vidutinių santykinių nuokrypių grafikas, kai įvedamas papildomas tabu koeficientas (ITP_11_3=0,5; ITP_11_4=0,95) irlyginamas su pradiniu testiniu pavyzdžiu



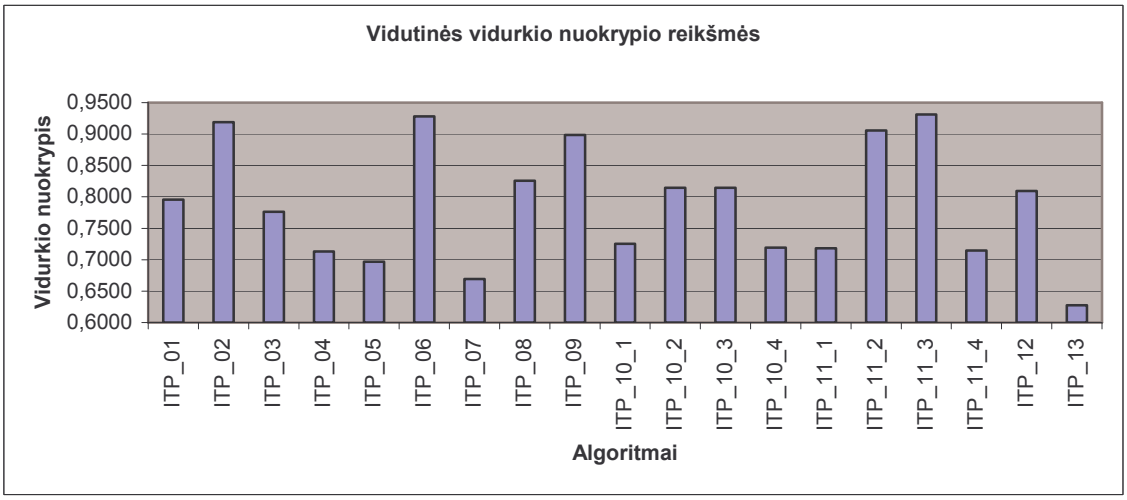
14 pav. Vidutinių santykinių vidurkio nuokrypių iliustravimas



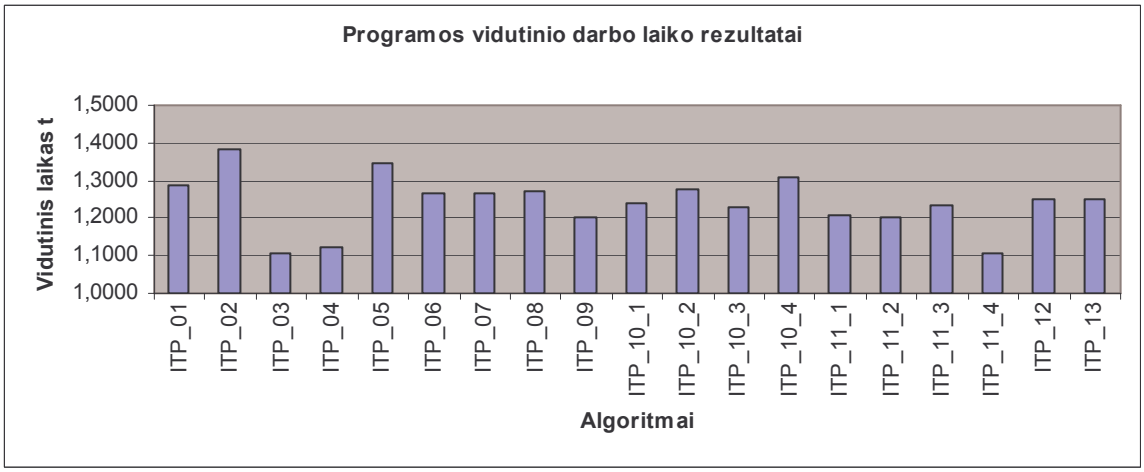
15 pav. Vidutinių santykinių nuokrypių grafikas, kai suvienodinamas iteracijų skaičius ir padvigubinamas pakartotinių paleidimų skaičius



16 pav. Vidutinių santykinių nuokrypių grafikas, kai per pusę sumažinmaas iteracijų skaičius ir padvigubinamas pakartotinių paleidimų skaičius



17 *pav. Vidutinių santykinių vidurkio nuokrypių iliustravimas*



18 pav. Vidutinio programos darbo laiko iliustravimas

6.4 Gautų rezultatų analizė

Pirmos tyrimo dalies metu buvo pakeistos mutavimo procedūros, kurios buvo realizuotos ITP_02- ITP_07 versijose (1-3 lentelės). Pagal gautus rezultatus matome, kad, vetinant vidutinio santykinio nuokrypio gautas reikšmes, geriausi rezultatai gauti naudojant postūmio mutaciją, kai segmentų skaičius yra lygus 5 (ITP_06). Nuo šios versijos rezultatų labai nedaug atsilieka ir išplėstinės inversijos ir įterpimo mutacijos (IIĮM, kai N=5) gautos reikšmės (ITP_04). Tačiau sugaišto vidutinio laiko atžvilgiu geriausi rezultatai buvo gauti panaudojant inversijos ir įterpimo mutaciją, kai segmentų skaičius lygus 3 (ITP_02). Todėl tolimesni tyrimai buvo atliekami naudojant inversijos ir įterpimo mutaciją, keičiant 3 arba 5 segmentų dalį.

Tabu paieškos algoritmo metu keičiant atsitiktinių dydžių lygį gauti rezultatai parodyti 4 lentelėje (ITP_03, ITP_08, ITP_09). Pagal šiuos rezultatus matome, kad geriausi rezultatai yra gauti, kai atsitiktinumo lygio dydžio koeficientas yra vos mažesnis už 1 (ITP_03).

5-8 lentelėse pateikti tyrimo rezultatai, kurių metu buvo nustatoma, kokiam papildomo tabu koeficiento dydžiui gaunami geriausi rezultatai. Tyrimo metu buvo lyginami rezultatai, esant IĮM, kai segmentų skaičius lygus 3 ir 5 (ITP_11 ir ITP_10). Pagal gautus rezultatus galima teigti, kad stagnacijos minimizavimas rezultatams labai didelės įtakos neturi, tačiau pagerėjimas geriausiai matosi esant inversijos ir įterpimo mutacijai, kai segmentų skaičius lygus 3. Kai kurių rezultatų palyginimas parodytas 13 pav.

9 lentelėje esantys rezultatai parodo, kad padidinus uždraudimų (tabu) sąrašo dydį, gauti vidutinio santykinio nuokrypio rezultatai yra geresni net 21,15% už pradinės versijos(ITP_01) rezultatus. Laiko atžvilgiu rezultatai pagerinami atitinkamai 13,85%.

Bendri eksperimentų rezultatai (iš 1-9 lentelių) pateikti 17-18 pav.

Paskutinis eksperimentas buvo skirtas nustatyti kokią įtaką daro iteracijų bei pakartotinių paleidimų skaičius (14 pav.). Pagal gautus rezultataus matome (10 lentelė), kad visiems testiniams pavyzdžiams suvienodinus iteracijų skaičių, gaunami dvigubai blogesni rezultatai vidutinio santykinio nuokrypio atžvilgiu, ir dvigubai geresni rezultatai programos darbo laiko atžvilgiu (ITP_14) (15 pav.). Padvigubinus pakartotinio paleidimo skaičių (ITP_15), programos darbo laikas pailgėjo 2,93 karto, tačiau vidutinio santykinio nuokrypio rezultatai sumažėjo 2,26 karto. Paskutinis eksperimentas buvo vykdomas, kai iteracijų skaičius sumažinamas per pusę, pakartotinio paleidimo skaičius paliekamas toks pat (ITP_16). Palyginus su ITP_14 versijos rezultatais programos darbo laikas padidėjo 1,67

karto, o vidutinio santykinio nuokrypio reikšmė sumažėjo iki 0,89 karto. Pagal gautus rezultatus galime daryti išvadą, kad pakartotinio paleidimo skaičiaus padidinimas turi didelę įtaką rezulatams. Todėl įverinus vidutinio santykinio nuokrypio ir darbo laiko rezultatų reikšmingumą, galima daryti išvadą, kad geresnis yra paskutinio tyrimo variantas (ITP_16), kurio metu programos darbas vyksta ilgiau, tačiau gaunami geresni vidutinio santykinio nuokrypio rezultatai (16 pav.). Aukštesnės kokybės rezultatai, nors ir pasiekti per šiek tiek ilgesnį laiką, vis tik yra geriau negu blogesnės kokybės rezultatai, gauti per trumpesnį laiką.

IŠVADOS

I. Šiame darbe nagrinėjamas modifikuotas iteratyviosios tabu paieškos (ITP) algoritmas vienam iš sudėtingų kombinatorinio optimizavimo uždavinių − komivojažieriaus uždaviniui. Žinoma, kad komivojažieriaus uždavinys yra NP-sunkus, ir efektyvių euristinių algoritmų šiam uždaviniui kūrimas vis dar tebėra aktuali ir daugelio tyrinėtojų nagrinėjama tema. Pasiūlytos ITP algoritmo modifikacijos ir yra bandymas prisidėti tobulinant esamus ir konstruojant naujus algoritmus šiam uždaviniui.

II. Pagrindinė modifikuoto iteratyviosios tabu paieškos algoritmo idėja yra ta, kad yra siekiama pagerinti įprastinę tabu paieškos veikimo schemą, panaudojant tam tikrus papildomus sprendinių pertvarkymus (mutacijas) bei parenkant kiek įmanoma optimalesnes parametrų reikšmes.

III. Pagrindinis mutavimo tikslas yra tam tikru mastu transformuoti lokaliai optimalius sprendinius − jų perdaug „nesugriaunant“ − ir kartojant tabu paiešką nuo tų rekonstruotų sprendinių apimti kiek įmanoma didesnius sprendinių „poerdvius“, taip sudarant palankesnes sąlygas artėti prie globaliai optimalių sprendinių.

IV. ITP algoritmo modifikacijos buvo lygintos su pradine (bazine) ITP algoritmo versija. Modifikacijos buvo išbandytos su įvairių tipų komivojažieriaus uždavinio testiniais pavyzdžiais (duomenimis) iš komivojažieriaus uždavinio duomenų bibliotekos TSPLIB. Kaip parodė eksperimentų rezultatai, pasiūlytos algoritmo modifikacijos įgalino pasiekti geresnės kokybės sprendinius, mažiau nutolusius nuo globaliojo optimumo.

V. Net ir esant pasiektiems geros kokybės rezultatams, tikslinga ir toliau ieškoti kitų ITP patobulinimo būdų. Galimi patobulinimai galėtų būti susiję su kitokio tipo mutavimo operatorių pritaikymu. Taip pat būtų tikslinga išbandyti pasiūlytą algoritmą, sprendžiant kitus uždavinius.

LITERATŪRA

[1] **Aarts E.H.L., Lenstra J.K. (eds.).** *Local Search in Combinatorial Optimization*, Chichester: Wiley, 1997.

[2] **Aarts E.H.L., Korst J.H.M., van Laarhoven P.J.M..** Simulated annealing. E.Aarts, J.K.Lenstra (red.), *Local Search in Combinatorial Optimization*, Wiley, Chichester, 1997, 91–120.

[3] **Battiti R., Tecchiolli G.** The reactive tabu search. *ORSA Journal on Computing*, 1994, vol.6, 126-140.

[4] **Blažauskas T., Blonskis J., Misevičius A., Smolinskas J.**. An overview of some heuristic algorithms for combinatorial optimization problems. *Informacinės technologijos ir valdymas*, 2004, Nr.1(30)

[5] **Cerný V.** A thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. Tech. Report, Comenius University, Bratislava, CSSR, 1982.

[6] **Cook W.,** Travelling Salesman Problem [interaktyvus]. 2007m. sausis. – [žiūrėta 2007-05-02]. Prieiga per internetą: <http://www.tsp.gateech.edu/>

[7] **Dizdarevic S., Inza I., Kuijpers C.M.H., Larranaga P., Murga R.H.** Genetic algorithms for the traveling salesman problem: a review of representations and operators. *Artificial Intelligence Review*, 1999, vol.13, 129-170.

[8] **Eimontienė I., Misevičius A.** Iteratyvioji tabu paieška ir jos modifikacijos komivojažieriaus uždaviniui, Respublikinės konferencijos „Informacinės technologijos 2007" pranešimų medžiaga, Kaunas, Technologija, 2007, 462-466.

[9] **Freisleben B., Merz P.,** A genetic local search algorithm for solving symmetric and a symmetric traveling salesman problems. Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, Nagoya, Japan, 1996, 616-621.

[10] **Garey M.R., Johnson D.S.** *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco: Freeman, 1979.

[11] **Glover F., Laguna M.** Tabu search. Kluwer, 1997.

[12] **Glover F**., Tabu Search: A Tutorial, *Interfaces*, 1990, 20(4):74-94.

[13] **Goldberg D. E.,** Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[14] **Golden B., Pepper J., Wasil E.** Solving the traveling salesman problem with annealing-based heuristics: a computa-tional study. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 2002, vol.32, 72-77.

[15] **Holland J.H.** *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[16] **Johnson D.S., McGeoch L.A..** The traveling salesman problem: a case study. In E.Aarts, J.K.Lenstra (eds.), *Local Search in Combinatorial Optimization*, Chichester: Wiley, 1997, 215-310.

[17] **Johnson D.S.** Local optimization and the traveling salesman problem. *In Proceedings of the 17th International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science*, Berlin: Springer, 1990, vol.443, 446-461.

[18] **Kirkpatrick S., Gelatt C.D., Jr.,Vecchi M.P.** Optimization by simulated annealing. *Science*, 1983, t.220, 671–680.

[19] **Lin S., Kernighan B.W..** An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 1973, vol.21, 498-516.

[20] **Lourenco, H.R.; Martin, O.; Stuzle, T.** Iterated local search. In: Glover, F.; Kochenberger G. (ed.), *Handbook of Metaheuristics*. Norwell: Kluwer, 2002, 321–353.

[21] **Metropolis N., Rosenbluth A., Rosenbluth M., Teller A.** Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, 1953, t.21, 1087–1092.

[22] **Misevičius A.**, Iteratyviosios Tabu paieškos algoritmas komivojažieriaus uždaviniui, Respublikinės konferencijos „Informacinės technologijos 2005" pranešimų medžiaga, Kaunas, Technologija, 2005, 91-101.

[23] **Misevičius A.**, Modernieji euristiniai optimizavimo algoritmai, Mokslinio-tiriamojo darbo (registr. nr. T-04078) ataskaita, Kaunas, 2004.

[24] **Misevičius A.** Ruin and recreate principle based approach for the quadratic assignment problem. *Lecture Notes in Computer Science*, 2003, vol.2723: *Genetic and Evolutionary Computation – GECCO 2003* (Chicago, USA), *Proceedings*, *Part I*, E.Cantú-Paz, J.A.Foster, K.Deb et al. (eds.), 598-609.

[25] **Reinelt G.** The traveling salesman: computational solutions for TSP applications. *Lecture Notes in Computer Science*, Berlin: Springer, 1994, vol.840.

[26] **Reinelt G.** TSPLIB – A traveling salesman problem library. *ORSA Journal on Computing*, 1991, vol.3-4, 376-385. Taip pat žr.: [interaktyvus]. 2004m. – [žiūrėta 2007-05-02]. Prieiga per internetą:
< http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ >

[27] **Papadimitriou C.H., Steiglitz K.,** *Combinatorial Optimization: Algorithms and Complexity*, Englwood Cliffs: Prentice-Hall, 1982.

SUMMARY

ITERATED TABU SEARCH AND ITS MODIFICATIONS FOR THE TRAVELLING SALESMAN PROBLEM

The main goal of this Master thesis is to implement and test the iterative tabu search algorithm and its modifications for a well-known combinatorial optimization problem, the travelling salesman problem (TSP).

The definition of the travelling salesman problem is as follows. We have the matrix $\boldsymbol{D} = (d_{ij})_{n\times n}$ of distances between cities and the set Π of all possible permutations of natural numbers from 1 to n. The aim is to find the best permutation $p_{\text{opt}} \in \Pi$ such that $p_{\text{opt}} = \arg\min_{p\in\Pi} z(p)$, where z is the objective function:

$$z(p) = \sum_{i=1}^{n-1} d_{p(i)p(i+1)} + d_{p(n)p(1)};$$

where p is the TSP-solution; n is the size of the problem. Permutations are usually called tours. The pairs $p(1), p(2))$, ..., $(p(i), p(i+1))$, ..., $(p(n), p(1))$ are called the edges. An element $j=p(i)$ denotes city j to visit at step i. Thus, solving TSP means searching the shortest closed tour, in which every city is visited only once.

TSP is NP-hard problem and still remains a great challenge for the researchers. The TSP also serves as an experimental basis for the investigation of various optimization techniques. Since there are no efficient exact algorithms for this problem, heuristic methods are often applied.

In this work, one of the heuristic algorithm – the iterated tabu search and its modifications are discussed. The work is organized as follows. Firstly, some basic definitions and preliminaries are given. Then, the iterated tabu search algoritm and its variants based on special type mutations are considered in more details. The ITS algorithms modifications were tested on the TSP instances from the TSP library TSPLIB. The results of this tests (experiments) are presented as well. The work is completed with the conclusions.

SANTRUMPŲ IR TERMINŲ ŽODYNĖLIS

AM – Atkaitinimo modeliavimas

EA – Euristinis algoritmas

GA – Genetiniai algoritmai

GO – Globalusis optimumas

IIĮM – Išplėstinė inversijos ir įterpimo mutacija

IĮM – Inversijos ir įterpimo mutacija

ILP – Iteratyvioji lokalioji paieška

ITP – Iteratyvioji tabu paieška

KO – Kombinatorinis optimizavimas

KU – Komivojažieriaus uždavinys

LO – Lokalusis optimumas

LP – Lokalioji paieška

TF – Tikslo funkcija

TP – Tabu paieška

TSPLIB – KU testinių pavyzdžių elektroninė biblioteka (angl. *Travelling Salesman Problem Library*)

PRIEDAI

Priedas A:

Trumpas vartotojo vadovas

Programos paleidimas ir parametrai

Programa yra pritaikyta DOS terpei, taip pat gali pilnai funkcionuoti ir Windows terpėje. Programai reikia nurodyti vieną pagrindinį parametrą: duomenų failo vardą. Turi būti išpildytos tokios sąlygos:

- Duomenų byla privalo būti patalpinta darbiniame programos kataloge. Pvz. duomenų byla *„burma14.dat"* yra pagrindiniame programos kataloge, tada programa paleidžiama taip: `optitsp.exe` *`burma14.dat`*
- Bylos pavadinime privalo būti bylos plėtinys *`.dat`*. Optimali tikslo funkcijos reikšmė turi būti nurodyta to paties pavadinimo byloje, kurios plėtinys *`.bkv`*.
- Duomenų bylos turinys turi būti korektiškas (turinys atitinka atitinkamos TSPLIB bylos turinį).
- Jeigu byloje trūksta kokio nors elemento arba yra kokių nors kitų klaidų, ekrane pateikiamas pranešimas.

Programos vykdymas

Vykdant programą ekrane bus matomas bylos vardas, galimų iteracijų skaičius, esamų iteracijų ir naujų paleidimų skaičius, bendras keitimų skaičius ir geriausia tikslo funkcijos reikšmė. Esant dideliems objektų skaičiams (n), paieška gali užtrukti.

Algoritmo vykdymą galima bet kada nutraukti klavišo F10 paspaudimu. Toks veiksmas nutrauktų algoritmą esamos iteracijos pabaigoje ir tai atsispindėtų rezultatų byloje.

Parametrų nustatymų (konfiguracijos) byla

Nustatymų byloje *optitsp.cfg* yra tokia pagrindinė informacija:

- Iteracijų kiekis (ITER_1) – sprendinio gerinimo procedūros paleidimų kiekis;
- Paleidimų kiekis (RS) – algoritmo paleidimų kiekis;
- Laiko apribojimas (RIMELIMIT) – programos vykdymo laiko apribojimas sekundėmis. Jei lygus 0, tai reiškia, kad laikas neribojamas;
- RANDSEED – naudojamas pradiniui sprendiniui sugeneruoti.

Nustatymų bylą galima koreguoti bet kokiu teksto redaktoriumi, bet negalima keisti jo pavadinimo ir parametrų pavadinimų, galima koreguoti tik parametrų reikšmes. Parametrų eiliškumas įtakos neturi.

Priedas B:

Parametrų nustatymų (konfiguracijos) bylos *optitsp.cfg* pavyzdys:

```
*------------------------------------------------------------------------*
* CONFIGURATION FILE OF THE OPTIMIZER FOR THE TRAVELLING SALESMAN PROBLEM *
*------------------------------------------------------------------------*
...
*--- main optimization parameters ---
ITER_1=450          * # of iterations (1) [1..10000000]
ITER_2=1            * # of iterations (1) [1..10000000]
N_FACT=0.660        * N-factor [>0]
NS_FACT=0.130       * NS-factor [0..1]
RS=1                * # of restarts [1..1000]
MS=10                * # of multi-starts [1..3000]
TIMELIMIT=0         * optimization time limit [>=0]
*--- auxiliary coefficients ---
TABU_VAR=1          * tabu search variant [1/2]
LTLS_FACT=0.100     * lower tabu list size factor [>=0]
HTLS_FACT=0.140     * higher tabu list size factor [>=0]
MUT_VAR=7           * mutation variant [1/2/3/4/5/6/7]
...
REST_VAR=1          * restart variant [1/2/3/4/5/6]
ESP_FACT=0.025      * expected stagnation period factor [>0]
...
RANDSEED=0123456789 * random number generator's seed
...
PAP_KOEF_TABU=0.095 * Tabu stagnation koef [0..0,5]
* end
*------------------------------------------------------------------------*
```

Priedas C:

ITP algoritmo išeities teksto byla „*IterTS1.pas*”:

```
{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *}
{* ITERTS1                                                                *}
{* ITERATED TABU SEARCH FOR THE TSP (VARIANT 1) (C) A. MISEVICIUS         *}
{* CREATED: 2005.10.22                                                    *}
{* (C) A.MISEVICIUS                                                       *}
{* Modified: 2007.03.14                                                   *}
{* I.Eimontiene                                                           *}
{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *}
unit IterTS1; { iterated tabu search procedure (1) }
interface
{$I header.dcl} { unit declarations header }
procedure iterated_tabu_search_1(restart_number:integer; iter_n_1:longint;
                                 iter_n_2,iter_n_3:integer; iter_n_4:longint;
                                 its_seed:longint; var rc:byte);
implementation
uses TwoOPt,   { 2-opt procedure }
     EGDesc,   { enhanced greedy descent procedure }
     RPGen,    { random permutation (solution) generator }
     Mutation, { permutation mutation procedure }
     MutInVer, { }
     Merge,    { merging procedure }
     NNReCon,  { nearest neighbour reconnect procedure }
     InsReCon, { insert-reconnect procedure }
     IDReCon,  { insert-descent-reconnect procedure }
     Objectiv, { objective function value }
     (*InterAct, { interactive tools }*)
     InfoLine, { information line(s) }
     Protocol, { protocol procedures }
     Timer,    { time calculation }
     Utils;    { utilities }
procedure iterated_tabu_search_1(restart_number:integer; iter_n_1:longint;
                                 iter_n_2,iter_n_3:integer; iter_n_4:longint;
                                 its_seed:longint; var rc:byte);
{ iterated tabu search procedure for the TSP (variant 1) }
{ restart_number - current restart number }
{ iter_n_1,iter_n_2,iter_n_3,iter_n_4 - numbers (limits) of iterations }
{ its_seed - random number generator's seed for its }
```

```
{ rc - return code }
{ common data: N - number of objects (cities) }
{              D - distance matrix            }
{ results: P - permutation (solution)   }
{          F - objective function value }
var   index_1: longint; { main cycle index (1) }
      index_2: integer; { main cycle index (2) }
      index_3: integer; { main cycle index (3) }
      i,j: integer;     { indexes }
      k: longint;
      u: integer;
      iter_n_3_stag : double; {***}
const delta: longint = 0;             { objective function difference }
      min_delta: longint = 0;         { minimum objective function difference}
      random_level: double = 0.95;    { tabu criterion randomization level }
      m1: longint = 0; m2: longint = 0;  m3: longint = 0; { mutation levels }
      min_delta_aspired: longint = 0; { auxiliary variables-constants }
      min_i: integer = 0; min_j: integer = 0;
      min_i_1: integer = 0; min_j_1: integer = 0;
      p_i: integer = 0; p_j: integer = 0; p_j_1: integer = 0;
      d_p_i_1_p_i: longint = 0;
      forbidden: boolean = FALSE;
      aspired: boolean = FALSE;
      improved: boolean = FALSE;
      tabu_i_j: integer = 0;
      stagnation_interval: integer = 0;
      min_index_1: longint = 0;
      old_index_3: integer = 0;
      local_opt_index: integer = 0;
      h: integer = 0;
      v: integer = 0;
      rp1: integer = 0; rp2: integer = 0;
      p_1_i: integer = 0; p_1_i_1: integer = 0;
      p_index_2_p_1_i: integer = 0;
      its_seed_1: longint = 0; its_seed_2: longint = 0;
      its_seed_3: longint = 0;
      aux_its_seed: longint = 0;
procedure initialize_tabu_list; { initialization of tabu matrix }
var  q: longint;
begin
     Current_Rand_Seed:=its_seed_1;
     Tabu_List_Size:=randint(Lower_Tabu_List_Size,Higher_Tabu_List_Size);
     its_seed_1:=Current_Rand_Seed;
     for q:=1 to Stack_Header do Tabu_List^[Stack_1^[q]]^[Stack_2^[q]]:=0;
     Stack_Header:=0;
end; { initialize_tabu_list }
procedure reinitialize_tabu_list(w:integer);
{ reinitialization of tabu matrix }
var  q: integer;
begin
     initialize_tabu_list; { initialization of tabu matrix }
     for q:=1 to w do begin
       rp1:=Rand_P[q]; rp2:=Rand_P[succ(q)];
       if rp1<rp2 then begin
            Tabu_List^[rp1]^[rp2]:=NOT_ALLOWED;
            Stack_Header:=succ(Stack_Header);
            Stack_1^[Stack_Header]:=rp1;
            Stack_2^[Stack_Header]:=rp2
          end
          else begin
            Tabu_List^[rp2]^[rp1]:=NOT_ALLOWED;
            Stack_Header:=succ(Stack_Header);
            Stack_1^[Stack_Header]:=rp2;
            Stack_2^[Stack_Header]:=rp1
          end
     end;
end; { reinitialize_tabu_list }
procedure add_to_tabu_list(item1,item2,item3,item4:integer);
{ procedure for adding items to tabu list }
begin
     Tabu_List^[item1]^[item2]:=index_3+Tabu_List_Size;
     Stack_Header:=succ(Stack_Header);
     Stack_1^[Stack_Header]:=item1;
     Stack_2^[Stack_Header]:=item2;
     Tabu_List^[item3]^[item4]:=index_3+Tabu_List_Size;
     Stack_Header:=succ(Stack_Header);
     Stack_1^[Stack_Header]:=item3;
```

```
     Stack_2^[Stack_Header]:=item4;
end; { add_to_tabu_list }
function distance(var p_1,p_2: fixed_int_array): integer;
{ distance between permutations p_1 and p_2 }
var   i: integer;
const dist: integer = 0;
begin
     dist:=0;
     p_1[Succ_N]:=p_1[1];
     p_2[0]:=p_2[N]; p_2[Succ_N]:=p_2[1];
     for i:=1 to N do P_Index_2[p_2[i]]:=i;
     for i:=1 to N do begin
       p_1_i:=p_1[i];
       p_1_i_1:=p_1[succ(i)];
       p_index_2_p_1_i:=P_Index_2[p_1_i];
       if ((p_1_i=p_2[p_index_2_p_1_i]) and
           ((p_1_i_1=p_2[succ(p_index_2_p_1_i)]) or
            (p_1_i_1=p_2[pred(p_index_2_p_1_i)]))) then { nothing }
                                                 else dist:=succ(dist)
     end;
     distance:=dist;
end; { distance }
procedure save_minimum; { saving the best local minimum }
begin
     if F<Best_F then begin
        Best_F:=F; { save the best objective function value }
        move(P[1],Best_P[1],N_Intsize); { save the best permutation }
        info_line(OBJ) { display objective function value }
     end;
end; { save_minimum }
begin { iterated_tabu_search_1 }
     iter_n_3_stag := iter_n_3*pap_koef_tabu;
     Min_F:=F; { minimum objective function value }
     move(P[1],Min_P[1],N_Intsize);
     Local_Opt_F_3:=F;
     m2:=pred(Min_M_Lev_2); { m2 - current mutation level (2) }
     m3:=pred(Min_M_Lev_3); { m3 - current mutation level (3) }
     min_index_1:=0;
     save_minimum; { save current solution }
     its_seed_1:=its_seed; its_seed_2:=succ(its_seed); its_seed_3:=its_seed;
     aux_its_seed:=MAX_MS_N+its_seed+
                   (longint(restart_number)-1+SeedOffset)*(MAX_ITER_N_1-1)-1;
     Current_Rand_Seed:=its_seed_1;
     Tabu_List_Size:=randint(Lower_Tabu_List_Size,Higher_Tabu_List_Size);
     its_seed_1:=Current_Rand_Seed;
     Stack_Header:=0;
     stagnation_interval:=maxi(1,round(iter_n_3*0.2));
     Current_N:=Pred_N;
     i:=2; j:=2;
     Tabu_List_I_:=@Tabu_List^[2]^[0];
     I1_:=2; J1_:=2;
     I2_:=2; J2_:=3; K2_:=4;
     I3_:=2; J3_:=3; K3_:=5; L3_:=6;
     I4_:=2; J4_:=3; K4_:=5; L4_:=7; M4_:=8;
     Variant_:=0;
     for index_1:=1 to iter_n_1 do begin { main cycle of its }
       if ELS=1 then begin enhanced_greedy_descent(NS_div_4{,its_seed_1});
          if F<Local_Opt_F_3 then begin
             Local_Opt_F_3:=F;
             if F<Min_F then begin
                Min_F:=F;
                move(P[1],Min_P[1],N_Intsize);
                save_minimum;
                m3:=pred(Min_M_Lev_3); { reset mutation level (3) }
             end;
             m2:=pred(Min_M_Lev_2); { reset mutation level (2) }
             min_index_1:=index_1
          end
       end; { ELS=1 }
       Local_Opt_F_2:=F;
       m1:=pred(Min_M_Lev_1);
       for index_2:=1 to iter_n_2 do begin
         Local_Opt_F_1:=F;
         old_index_3:=1;
         local_opt_index:=1;
         index_3:=1;
         improved:=FALSE;
```

```
while ((index_3<=iter_n_3) or improved) do begin
  if index_3-local_opt_index>stagnation_interval then begin
     mutation_4(TABU,1,VARIANT_1,VARIANT_4,index_3,its_seed_1);
     local_opt_index:=index_3
  end;
  min_delta:=INFINITY;
  p_i:=P[i];
  D_P_I_:=@D^[p_i]^[0];
  D_P_I_1_:=@D^[P[pred(i)]]^[0];
  d_p_i_1_p_i:=D_P_I_1_^[p_i];
  for k:=1 to iter_n_4 do begin
    if j<Current_N then j:=succ(j)
       else begin
         if i<Pred_N then begin
                       if i<>2 then { nothing }
                              else Current_N:=N;
                       i:=succ(i)
                     end
                     else begin
                       i:=2;
                       Current_N:=Pred_N
                     end;
         j:=succ(i);
         p_i:=P[i];
         D_P_I_:=@D^[p_i]^[0];
         D_P_I_1_:=@D^[P[pred(i)]]^[0];
         d_p_i_1_p_i:=D_P_I_1_^[p_i];
         Tabu_List_I_:=@Tabu_List^[i]^[0]
       end;
    tabu_i_j:=Tabu_List_I_^[j];
       p_j:=P[j];
       p_j_1:=P[succ(j)];
       delta:=D_P_I_1_^[p_j]+D_P_I_^[p_j_1]-
              d_p_i_1_p_i-D^[p_j]^[p_j_1];
       forbidden:=((tabu_i_j>=index_3) and
                   (randdouble<random_level));
       aspired:=(F+delta<Local_Opt_F_1);
       if ((delta<min_delta) and
          ((not forbidden) or
          ((aspired) and (index_3>iter_n_3_stag)))) then begin
          min_delta:=delta;
          min_i:=i; min_j:=j
       end
  end; { for k }
  if min_delta<>INFINITY then begin
       F:=F+min_delta; { new objective function value }
       improved:=(min_delta<0);
       h:=pred(min_i+min_j) shr 1;
       v:=min_j;
       for u:=min_i to h do begin
         p_i:=P[u]; P[u]:=P[v]; P[v]:=p_i;
         v:=pred(v)
       end;
       if F<Local_Opt_F_1 then begin
          Local_Opt_F_1:=F;
          local_opt_index:=index_3;
          if F<Local_Opt_F_2 then begin
             Local_Opt_F_2:=F;
             m1:=pred(Min_M_Lev_1);
             if F<Local_Opt_F_3 then begin
                Local_Opt_F_3:=F;
                if F<Min_F then begin
                   Min_F:=F;
                   move(P[1],Min_P[1],N_Intsize);
                   save_minimum;
                   m3:=pred(Min_M_Lev_3); { reset mutation level (3) }
                end;
                m2:=pred(Min_M_Lev_2); { reset mutation level (2) }
                min_index_1:=index_1
             end
          end
       end;
       add_to_tabu_list(pred(min_i),min_j,min_i,succ(min_j));
       Interchanges_Count:=succ(Interchanges_Count)
    end { min_delta<>INFINITY }
    else improved:=FALSE;
  if index_3 mod Higher_Tabu_List_Size_2 <> 0 then { nothing }
```

```
      else begin
        Current_Rand_Seed:=its_seed_1;
        Tabu_List_Size:=randint(Lower_Tabu_List_Size,
                                Higher_Tabu_List_Size);
        its_seed_1:=Current_Rand_Seed
      end;
    if (improved and (longint(index_3)-old_index_3>=
                    AIF_Fact*Tabu_List_Size)) then begin
        two_opt(index_3,its_seed_1);
        if F<Local_Opt_F_1 then begin
           Local_Opt_F_1:=F;
           local_opt_index:=index_3;
           if F<Local_Opt_F_2 then begin
              Local_Opt_F_2:=F;
              m1:=pred(Min_M_Lev_1);
              if F<Local_Opt_F_3 then begin
                 Local_Opt_F_3:=F;
                 if F<Min_F then begin
                    Min_F:=F;
                    move(P[1],Min_P[1],N_Intsize);
                    save_minimum;
                    m3:=pred(Min_M_Lev_3); { reset mutation level (3) }
                 end;
                 m2:=pred(Min_M_Lev_2);
                 min_index_1:=index_1
              end
           end
        end;
        if index_3>=iter_n_3 then break;
        mutation_4(TABU,maxi(1,round(N*0.01)),VARIANT_30,VARIANT_232,
                   index_3,its_seed_1);
        old_index_3:=index_3;
        improved:=FALSE
      end
      else index_3:=succ(index_3)
  end; { while }
  if index_2=iter_n_2 then break;
  if m1<Max_M_Lev_1 then m1:=succ(m1)
                    else m1:=Min_M_Lev_1;
  move(Min_P[1],P[1],N_Intsize);
  case MutationVariant of
    0:;{ nothing }
    1: begin
        mutation_1(P,m1,its_seed_2);
        calculate_objective_function_value;
        reinitialize_tabu_list(pred(m1))
      end; { 1 }
    2: begin
        F:=Min_F;
        mutation_2(P,F,m1,its_seed_2);
        initialize_tabu_list
      end; { 2 }
    3: begin
        F:=Min_F;
        mutation_4(NOT_TABU,m1,VARIANT_30,VARIANT_232,0,its_seed_2);
        initialize_tabu_list
      end; { 3 }
    4: begin
        Fast_Calculation:=(m1<pred(N_div_2));
        if Fast_Calculation then F:=Min_F;
        nearest_neighbour_reconnect(m1,its_seed_2);
        if Fast_Calculation then { nothing }
           else calculate_objective_function_value;
        initialize_tabu_list
      end; { 4 }
    5: begin
        insert_reconnect(P,m1,1,1,its_seed_2);
        calculate_objective_function_value;
        initialize_tabu_list
      end; { 5 }
    6: begin
        insert_descent_reconnect(P,m1,1,1,its_seed_2);
        calculate_objective_function_value;
        initialize_tabu_list
      end; { 6 }
    7: begin
        inversion_mutation(P,m1, its_seed_2);
```

```
         calculate_objective_function_value;
         initialize_tabu_list
       end; { 7 }
    otherwise;
  end; { case }
  its_seed_2:=Current_Rand_Seed
end; { for index_2 }
if ELS=2 then begin
   enhanced_greedy_descent(NS_div_2);
   if F<Local_Opt_F_3 then begin
      Local_Opt_F_3:=F;
      if F<Min_F then begin
         Min_F:=F;
         move(P[1],Min_P[1],N_Intsize);
         save_minimum;
         m3:=pred(Min_M_Lev_3); { reset mutation level (3) }
      end;
      m2:=pred(Min_M_Lev_2); { reset mutation level (2) }
      min_index_1:=index_1
   end
end; { ELS=2 }
if index_1<iter_n_1 then begin
if succ(index_1-min_index_1)>=ES_Period then begin
   aux_its_seed:=succ(aux_its_seed);
   its_seed_1:=aux_its_seed; its_seed_2:=succ(aux_its_seed);
   its_seed_3:=aux_its_seed;
   case RestartVariant of
     0:;{ nothing }
     1: begin
          prot_text('+');
          if distance(Min_P,P)<(N div 4) then begin
               move(Min_P[1],P[1],N_Intsize);
               P[Succ_N]:=P[1];
               mutation_3(P,aux_its_seed);
               calculate_objective_function_value
            end
            else begin
              merge_permutations_2(Min_P,P,Temp_P,aux_its_seed);
              move(Temp_P[1],P[1],N_Intsize);
              P[Succ_N]:=P[1];
              calculate_objective_function_value
            end
        end; { 1 }
     2: begin
          { go to the next mutation level m3 }
          prot_text('+');
          if m3<Max_M_Lev_3 then m3:=succ(m3)
                            else m3:=Min_M_Lev_3;
          move(Min_P[1],P[1],N_Intsize);
          P[Succ_N]:=P[1];
          F:=Min_F;
          mutation_4(NOT_TABU,m3,VARIANT_30,VARIANT_232,0,aux_its_seed);
          enhanced_greedy_descent(NS*2);
          two_opt_;
          if F<Min_F then begin
             Min_F:=F;
             move(P[1],Min_P[1],N_Intsize);
             save_minimum;
             m3:=pred(Min_M_Lev_3);
          end
        end; { 2 }
     3: begin
          prot_text('+');
          if distance(Min_P,P)<(N div 4) then begin
               if m3<Max_M_Lev_3 then m3:=succ(m3)
                                 else m3:=Min_M_Lev_3;
               move(Min_P[1],P[1],N_Intsize);
               P[Succ_N]:=P[1];
               F:=Min_F;
               mutation_4(NOT_TABU,m3,VARIANT_30,VARIANT_232,
                          0,aux_its_seed)
             end
             else begin
               Merge_Block_Length:=maxi(1,mini(round(N*MBL_FACT_1),Pred_N));
               merge_permutations_2(Min_P,P,Temp_P,aux_its_seed);
               move(Temp_P[1],P[1],N_Intsize);
               P[Succ_N]:=P[1];
```

```
              calculate_objective_function_value
            end;
         if ELS=3 then begin
            enhanced_greedy_descent(NS*2);
            two_opt_;
            if F<Min_F then begin
               Min_F:=F;
               move(P[1],Min_P[1],N_Intsize);
               save_minimum;
               m3:=pred(Min_M_Lev_3);
            end
         end { ELS=3 }
      end; { 3 }
   4: begin
         prot_text('+');
         if m3<Max_M_Lev_3 then m3:=succ(m3)
                           else m3:=Min_M_Lev_3;
         move(Min_P[1],P[1],N_Intsize);
         Fast_Calculation:=(m3<pred(N_div_2));
         if Fast_Calculation then F:=Min_F;
         nearest_neighbour_reconnect(m3,aux_its_seed);
         if Fast_Calculation then { nothing }
            else calculate_objective_function_value;
         if ELS=3 then begin
            enhanced_greedy_descent(NS*2);
            two_opt_;
            if F<Min_F then begin
               Min_F:=F;
               move(P[1],Min_P[1],N_Intsize);
               save_minimum;
               m3:=pred(Min_M_Lev_3);
            end
         end { ELS=3 }
      end; { 4 }
   5: begin
         prot_text('+');
         if m3<Max_M_Lev_3 then m3:=succ(m3)
                           else m3:=Min_M_Lev_3;
         move(Min_P[1],P[1],N_Intsize);
         insert_reconnect(P,m3,20,30,aux_its_seed);
         calculate_objective_function_value;
         if ELS=3 then begin
            enhanced_greedy_descent(NS_div_8);
            if F<Min_F then begin
               Min_F:=F;
               move(P[1],Min_P[1],N_Intsize);
               save_minimum;
               m3:=pred(Min_M_Lev_3);
            end
         end { ELS=3 }
      end; { 5 }
   6: begin
         prot_text('+');
         if m3<Max_M_Lev_3 then m3:=succ(m3)
                           else m3:=Min_M_Lev_3;
         move(Min_P[1],P[1],N_Intsize);
         insert_descent_reconnect(P,m3,25,50,aux_its_seed);
         calculate_objective_function_value;
         if ELS=3 then begin
            enhanced_greedy_descent(NS_div_8);
            if F<Min_F then begin
               Min_F:=F;
               move(P[1],Min_P[1],N_Intsize);
               save_minimum;
               m3:=pred(Min_M_Lev_3);
            end
         end { ELS=3 }
      end; { 6 }
   7: prot_text('+');
   otherwise;
  end; { case }
  initialize_tabu_list;
  Local_Opt_F_3:=F;
  m2:=pred(Min_M_Lev_2);
  min_index_1:=index_1;
  continue
end;
```

```
if m2<Max_M_Lev_2 then m2:=succ(m2)
                  else m2:=Min_M_Lev_2;
if Mut_Iter_N<=1 then begin
     move(Min_P[1],P[1],N_Intsize);
     case MutationVariant of
       0:;{ nothing }
       1: begin
            mutation_1(P,m2,its_seed_3);
            calculate_objective_function_value
          end; { 1 }
       2: begin
            F:=Min_F;
            mutation_2(P,F,m2,its_seed_3)
          end; { 2 }
       3: begin
            F:=Min_F;
            mutation_4(NOT_TABU,m2,VARIANT_30,VARIANT_232,0,its_seed_3)
          end; { 3 }
       4: begin
            Fast_Calculation:=(m2<pred(N_div_2));
            if Fast_Calculation then F:=Min_F;
            nearest_neighbour_reconnect(m2,its_seed_3);
            if Fast_Calculation then { nothing }
               else calculate_objective_function_value
          end; { 4 }
       5: begin
            insert_reconnect(P,m2,4,7,its_seed_3);
            prot_text('5 mutacija pavyko');
            calculate_objective_function_value
          end; { 5 }
       6: begin
            insert_descent_reconnect(P,m2,7,10,its_seed_3);
            calculate_objective_function_value
          end; { 6 }
       7: begin
            inversion_mutation(P,m2, its_seed_3);
            calculate_objective_function_value
          end; { 7 }
       otherwise;
     end; { case }
     its_seed_3:=Current_Rand_Seed
   end
   else begin
     Saved_F:=INFINITY;
     for k:=1 to Mut_Iter_N do begin
       move(Min_P[1],P[1],N_Intsize);
       P[Succ_N]:=P[1];
       case MutationVariant of
         0:;{ nothing }
         1: begin
              mutation_1(P,m2,its_seed_3);
              calculate_objective_function_value
            end; { 1 }
         2: begin
              F:=Min_F;
              mutation_2(P,F,m2,its_seed_3)
            end; { 2 }
         3: begin
              F:=Min_F;
              mutation_4(NOT_TABU,m2,VARIANT_30,VARIANT_232,0,its_seed_3)
            end; { 3 }
         4: begin
              Fast_Calculation:=(m2<pred(N_div_2));
              if Fast_Calculation then F:=Min_F;
              nearest_neighbour_reconnect(m2,its_seed_3);
              if Fast_Calculation then { nothing }
                 else calculate_objective_function_value
            end; { 4 }
         5: begin
              insert_reconnect(P,m2,4,7,its_seed_3);
              prot_text('5 mutacija pavyko');
              calculate_objective_function_value
            end; { 5 }
         6: begin
              insert_descent_reconnect(P,m2,7,10,its_seed_3);
              calculate_objective_function_value
            end; { 6 }
```

```
                7: begin
                     inversion_mutation(P,m1, its_seed_2);
                     calculate_objective_function_value
                   end; { 7 }
                otherwise;
              end; { case }
              its_seed_3:=Current_Rand_Seed;
              if F<Saved_F then begin
                 Saved_F:=F;
                 move(P[1],Saved_P[1],N_Intsize);
                 move(Rand_P[1],Temp_Rand_P[1],N_Intsize)
              end
            end; { for k }
            F:=Saved_F;
            move(Saved_P[1],P[1],N_Intsize);
            P[Succ_N]:=P[1];
            move(Temp_Rand_P[1],Rand_P[1],N_Intsize)
          end;
       case MutationVariant of
         0: initialize_tabu_list;
         1: reinitialize_tabu_list(pred(m2));
         2: initialize_tabu_list;
         3: initialize_tabu_list;
         4: initialize_tabu_list;
         5: initialize_tabu_list;
         6: initialize_tabu_list;
         7: initialize_tabu_list;
         otherwise;
       end { case }
       end { index_1<iter_n_1 }
     end; { main cycle }
     for k:=1 to Stack_Header do Tabu_List^[Stack_1^[k]]^[Stack_2^[k]]:=0;
     F:=Min_F;
end; { iterated_tabu_search_1 }
end. { iterts1 }
```

Priedas D:

Mutavimo procedūrų išeities bylos „*MutInver.pas*” (ITP_02, ITP_03, ITP_04, ITP_05, ITP_06, ITP_07):

ITP_02

```
{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *}
{* MUTINVER                                                                 *}
{* MUTATION AND INVERSION PROCEDURES                                        *}
{* CREATED: 2006.12.21                                                      *}
{* (C) I. EIMONTIENE                                                        *}
{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *}
unit MutInver; { inversion mutation procedures }
interface
{$I header.dcl} { unit declarations header }
procedure inversion_mutation(var current_p:fixed_int_array; m:integer; seed:longint);
implementation
uses RPGen,    { random permutation generator }
     Update1,  { procedure for updating permutation (1) }
     Update2,  { procedure for updating permutation (2) }
     Update31, { procedure for updating permutation (3.1) }
     Update32, { procedure for updating permutation (3.2) }
     Update33, { procedure for updating permutation (3.3) }
     Update34, { procedure for updating permutation (3.4) }
     Update35, { procedure for updating permutation (3.5) }
     Update36, { procedure for updating permutation (3.6) }
     Update37, { procedure for updating permutation (3.7) }
     Protocol, { protocol procedures }
     Utils;    { utilities }
procedure inversion_mutation(var current_p:fixed_int_array; m:integer; seed:longint);
{ procedure for the Inversion mutation (IVM) }
{ current_p - current permutation }
{ m - mutation level }
{ seed - random number generator's seed for mutated permutation }
const m_1: integer = 0;
      ri: integer = 0; ri1: integer = 0;
      ri_inv: integer = 0;
      sk: integer = 1;   {išėmimo skaičius}
```

```
      beg: integer = 0;
var   i: integer; { index }
      ii: integer; { index }
      t:  integer;     {išimamas kelias}
begin { simple_mutation_inv, sk=1 }
  current_p[Succ_N]:=current_p[1];
  Current_Rand_Seed:=seed;
  move(current_p[1],Rand_P[1],N_Intsize);
  ri := randint(1,N);
  t := Rand_P[ri];
  ii:=0;
  for i:=ri to N-1 do begin {išmetama nereikalinga dalis}
    Rand_P[i]:=Rand_P[i+1]
  end;
  ri_inv := randint(1,N-1);
  ii:= N-1;
  while ii>ri_inv-1 do begin
    Rand_P[ii+1] := Rand_P[ii];
    ii:=pred(ii)
  end;
  Rand_P[ri_inv] := t;
  move(Rand_P[1],current_p[1],N_Intsize);
  current_p[Succ_N]:=current_p[1];
end; { inversion_mutation }
end. { mutation and inversion }
```

ITP_03(sk=3), **ITP_04** (sk=5)

```
{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *}
{* MUTINVER                                                                    *}
{* MUTATION AND INVERSION PROCEDURES                                           *}
{* CREATED: 2006.11.28                                                         *}
{* (C) I. EIMONTIENE                                                           *}
{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *}
unit MutInver; { inversion mutation procedures }
interface
{$I header.dcl} { unit declarations header }
procedure inversion_mutation(var current_p:fixed_int_array; m:integer; seed:longint);
implementation
uses RPGen,    { random permutation generator }
     Update1,  { procedure for updating permutation (1) }
     Update2,  { procedure for updating permutation (2) }
     Update31, { procedure for updating permutation (3.1) }
     Update32, { procedure for updating permutation (3.2) }
     Update33, { procedure for updating permutation (3.3) }
     Update34, { procedure for updating permutation (3.4) }
     Update35, { procedure for updating permutation (3.5) }
     Update36, { procedure for updating permutation (3.6) }
     Update37, { procedure for updating permutation (3.7) }
     Protocol, { protocol procedures }
     Utils;    { utilities }
procedure inversion_mutation(var current_p:fixed_int_array; m:integer; seed:longint);
{ procedure for the Inversion mutation (IVM) }
{ current_p - current permutation }
{ m - mutation level }
{ seed - random number generator's seed for mutated permutation }
const m_1: integer = 0;
      ri: integer = 0; ri1: integer = 0;
      ri_inv: integer = 0;
      sk: integer = 3;   {išėmimo skaičius}
      beg: integer = 0;
var   i: integer; { index }
      ii: integer; { index }
      t: array[0..2] of integer;     {išimamas kelias}
      t_inv: array[0..2] of integer; {išimtas kelias jau apverstas}
begin { simple_mutation_inv,sk=3 }
  current_p[Succ_N]:=current_p[1];
  Current_Rand_Seed:=seed;
  move(current_p[1],Rand_P[1],N_Intsize);
  ri := randint(1,N);
  beg := 0;
  for i:=0 to sk-1 do begin
    if ri+i<=N then begin
      t[i+1] := Rand_P[ri+i]
    end
    else begin
      beg:=succ(beg);
```

```
      t[i+1] := Rand_P[beg]
    end;
  end;
  for i:=0 to sk-1 do begin {apverčiama - vykdoma inversija}
    t_inv[i+1] := t[sk-i]
  end;
  beg:=0;
  ii:=0;
  if ri+sk<=N+1 then begin
    for i:=ri to N-sk do begin {išmetama nereikalinga dalis}
      Rand_P[i]:=Rand_P[i+sk]
    end
  end
  else begin
    ii:=sk+ri-N-1;
    for i:=ri+sk-N to N-sk+ii do begin
      beg := succ(beg);
      Rand_P[beg]:=Rand_P[i]
    end
  end;
  N := N-sk;
  ri_inv := randint(1,N);
  i:=0;
  while i<sk do begin
    ii:= N+i;
    while ii>ri_inv-1 do begin
      Rand_P[ii+1] := Rand_P[ii];
      ii:=pred(ii)
    end;
    i:=succ(i)
  end;
  N := N+sk;
  i:=0;
  for ii:=ri_inv to ri_inv+sk-1 do begin
    i:= succ(i);
    Rand_P[ii] := t_inv[i]
  end;
  move(Rand_P[1],current_p[1],N_Intsize);
  current_p[Succ_N]:=current_p[1];
end; { inversion_mutation }
end. { mutation and inversion }
```

ITP_05

```
{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *}
{* MUTINVER                                                                   *}
{* MUTATION AND INVERSION PROCEDURES                                          *}
{* CREATED: 2006.11.06                                                        *}
{* (C) I. EIMONTIENE                                                          *}
{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *}
unit MutInver; { inversion mutation procedures }
interface
{$I header.dcl} { unit declarations header }
procedure inversion_mutation(var current_p:fixed_int_array; m:integer; seed:longint);
implementation
uses RPGen,    { random permutation generator }
     Update1,  { procedure for updating permutation (1) }
     Update2,  { procedure for updating permutation (2) }
     Update31, { procedure for updating permutation (3.1) }
     Update32, { procedure for updating permutation (3.2) }
     Update33, { procedure for updating permutation (3.3) }
     Update34, { procedure for updating permutation (3.4) }
     Update35, { procedure for updating permutation (3.5) }
     Update36, { procedure for updating permutation (3.6) }
     Update37, { procedure for updating permutation (3.7) }
     Protocol, { protocol procedures }
     Utils;    { utilities }
procedure inversion_mutation(var current_p:fixed_int_array; m:integer; seed:longint);
{ procedure for the Inversion mutation (IVM) }
{ current_p - current permutation }
{ m - mutation level }
{ seed - random number generator's seed for mutated permutation }
const m_1: integer = 0;
      sk: integer = 5;   {išėmimo skaičius}
      beg: integer = 0;
      kof: integer = 0; {ciklo koeficientas, priklausantis nuo miestu sk. N}
var   i: integer; { index }
```

```
      ii: integer; { index }
      j: integer;  { index }
      kof2: integer; {segmentu zingsnis}
      t: array[0..4] of integer;     {išimamas kelias}
      t_inv: array[0..4] of integer; {išimtas kelias jau apverstas}
      ri: array[0..MAX_N] of integer;
      ri_inv: array[0..MAX_N] of integer;
      pbb: integer; {ri rand pabaiga}
      zng: integer; {ri rand zingsnis}
      prr: integer;  {ri rand pradzia}
begin { simple_mutation_inv }
  kof:=N div sk;
  kof:= round(kof*0.5);
  if kof<1 then kof:=1;
  prr:=1;
  pbb:=0;
  zng:=N div kof;
  for j:=0 to kof-1 do begin
    prr:=pbb+1;
    pbb:=zng*(j+1);
    ri[j]:=randint(prr,pbb);
    ri_inv[j]:=randint(prr,pbb-sk);
  end;
  for j:=0 to kof-1 do begin
    current_p[Succ_N]:=current_p[1];
    Current_Rand_Seed:=seed;
    move(current_p[1],Rand_P[1],N_Intsize);
    beg := 0;
    for i:=0 to sk-1 do begin
      if ri[j]+i<=N then begin
        t[i+1] := Rand_P[ri[j]+i]
      end
      else begin
        beg:=succ(beg);
        t[i+1] := Rand_P[beg]
      end;
    end;
    for i:=0 to sk-1 do begin {apverčiama - vykdoma inversija}
      t_inv[i+1] := t[sk-i]
    end;
    beg:=0;
    ii:=0;
    if ri[j]+sk<=N+1 then begin
      for i:=ri[j] to N-sk do begin {išmetama nereikalinga dalis}
        Rand_P[i]:=Rand_P[i+sk]
      end
    end
    else begin
      ii:=sk+ri[j]-N-1;
      for i:=ri[j]+sk-N to N-sk+ii do begin
        beg := succ(beg);
        Rand_P[beg]:=Rand_P[i]
      end
    end;
    N := N-sk;
    i:=0;
    while i<sk do begin
      ii:= N+i;
      while ii>ri_inv[j]-1 do begin
        Rand_P[ii+1] := Rand_P[ii];
        ii:=pred(ii)
      end;
      i:=succ(i)
    end;
    N := N+sk;
    i:=0;
    for ii:=ri_inv[j] to ri_inv[j]+sk-1 do begin
      i:= succ(i);
      Rand_P[ii] := t_inv[i]
    end;
    move(Rand_P[1],current_p[1],N_Intsize);
    current_p[Succ_N]:=current_p[1]
  end;
end; { inversion_mutation }
end. { mutation and inversion }
```

ITP_06

```
{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *}
{* MUTINVER                                                                 *}
{* MUTATION AND INVERSION PROCEDURES                                        *}
{* CREATED: 2006.11.16                                                      *}
{* (C) I. EIMONTIENE                                                        *}
{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *}
unit MutInver; { inversion mutation procedures }
interface
{$I header.dcl} { unit declarations header }
procedure inversion_mutation(var current_p:fixed_int_array; m:integer;
seed:longint);
implementation
uses RPGen,    { random permutation generator }
     Update1,  { procedure for updating permutation (1) }
     Update2,  { procedure for updating permutation (2) }
     Update31, { procedure for updating permutation (3.1) }
     Update32, { procedure for updating permutation (3.2) }
     Update33, { procedure for updating permutation (3.3) }
     Update34, { procedure for updating permutation (3.4) }
     Update35, { procedure for updating permutation (3.5) }
     Update36, { procedure for updating permutation (3.6) }
     Update37, { procedure for updating permutation (3.7) }
     Protocol, { protocol procedures }
     Utils;    { utilities }
procedure inversion_mutation(var current_p:fixed_int_array; m:integer;
seed:longint);
{ procedure for the Inversion mutation (IVM) }
{ current_p - current permutation }
{ m - mutation level }
{ seed - random number generator's seed for mutated permutation }
const m_1: integer = 0;
      ri: integer = 0; ri1: integer = 0;
      ri_inv: integer = 0;
      sk: integer = 5;   {išėmimo skaičius}
      beg: integer = 0;
var   i: integer; { index }
      ii: integer; { index }
      t: array[0..4] of integer;     {išimamas kelias}
      t_inv: array[0..4] of integer; {išimtas kelias jau apverstas}
begin { simple_mutation_inv }
  current_p[Succ_N]:=current_p[1];
  Current_Rand_Seed:=seed;
  move(current_p[1],Rand_P[1],N_Intsize);
  ri := randint(1,N);
  beg := 0;
  for i:=0 to sk-1 do begin
    if ri+i<=N then begin
      t[i+1] := Rand_P[ri+i]
    end
    else begin
      beg:=succ(beg);
      t[i+1] := Rand_P[beg]
    end;
  end;
  for i:=0 to sk-1 do begin {apverčiama - vykdoma inversija}
    t_inv[i+1] := t[sk-i]
  end;
  beg := 0;
  for i:=0 to sk-1 do begin
    if ri+i<=N then begin
      Rand_P[ri+i] := t_inv[i+1]
    end
    else begin
      beg:=succ(beg);
      Rand_P[beg] := t_inv[i+1]
    end;
```

```
  end;
  move(Rand_P[1],current_p[1],N_Intsize);
  current_p[Succ_N]:=current_p[1];
end; { inversion_mutation }
end. { mutation and inversion }
```

ITP_07

```
{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *}
{* MUTINVER                                                                    *}
{* MUTATION AND INVERSION PROCEDURES                                           *}
{* CREATED: 2006.12.04                                                         *}
{* (C) I. EIMONTIENE                                                           *}
{* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *}
unit MutInver; { inversion mutation procedures }
interface
{$I header.dcl} { unit declarations header }
procedure inversion_mutation(var current_p:fixed_int_array; m:integer;
seed:longint);
implementation
uses RPGen,    { random permutation generator }
     Update1,  { procedure for updating permutation (1) }
     Update2,  { procedure for updating permutation (2) }
     Update31, { procedure for updating permutation (3.1) }
     Update32, { procedure for updating permutation (3.2) }
     Update33, { procedure for updating permutation (3.3) }
     Update34, { procedure for updating permutation (3.4) }
     Update35, { procedure for updating permutation (3.5) }
     Update36, { procedure for updating permutation (3.6) }
     Update37, { procedure for updating permutation (3.7) }
     Protocol, { protocol procedures }
     Utils;    { utilities }
procedure inversion_mutation(var current_p:fixed_int_array; m:integer;
seed:longint);
{ procedure for the Inversion mutation (IVM) }
{ current_p - current permutation }
{ m - mutation level }
{ seed - random number generator's seed for mutated permutation }
const m_1: integer = 0;
      ri: integer = 0; ri1: integer = 0;
      ri_inv: integer = 0;
      sk: integer = 5;   {išėmimo skaičius}
      beg: integer = 0;
var   i: integer; { index }
      ii: integer; { index }
      t: array[0..4] of integer;     {išimamas kelias}
      t_inv: array[0..4] of integer; {išimtas kelias jau apverstas}
begin { simple_mutation_inv }
  current_p[Succ_N]:=current_p[1];
  Current_Rand_Seed:=seed;
  move(current_p[1],Rand_P[1],N_Intsize);
  ri := randint(1,N);
  beg := 0;
  for i:=0 to sk-1 do begin
    if ri+i<=N then begin
      t[i+1] := Rand_P[ri+i]
    end
    else begin
      beg:=succ(beg);
      t[i+1] := Rand_P[beg]
    end;
  end;
  for i:=0 to sk-1 do begin {apverčiama - vykdoma inversija}
    t_inv[i+1] := t[i+1]
  end;
  beg:=0;
  ii:=0;
  if ri+sk<=N+1 then begin
```

```
    for i:=ri to N-sk do begin {išmetama nereikalinga dalis}
      Rand_P[i]:=Rand_P[i+sk]
    end
  end
  else begin
    ii:=sk+ri-N-1;
    for i:=ri+sk-N to N-sk+ii do begin
      beg := succ(beg);
      Rand_P[beg]:=Rand_P[i]
    end
  end;
  N := N-sk;
  ri_inv := randint(1,N);
  i:=0;
  while i<sk do begin
    ii:= N+i;
    while ii>ri_inv-1 do begin
      Rand_P[ii+1] := Rand_P[ii];
      ii:=pred(ii)
    end;
    i:=succ(i)
  end;
  N := N+sk;
  i:=0;
  for ii:=ri_inv to ri_inv+sk-1 do begin
    i:= succ(i);
    Rand_P[ii] := t_inv[i]
  end;
  move(Rand_P[1],current_p[1],N_Intsize);
  current_p[Succ_N]:=current_p[1];
end; { inversion_mutation }
end. { mutation and inversion }
```

Priedas E:

Straipsnis, atspausdintas Kauno tecnologijos universiteto išleistame leidinyje „Informacinės technologijos 2007, Respublikinės konferencijos pranešimų medžiaga" Kaunas, Technologija, 2007, 462 – 466.

ITERATYVIOJI TABU PAIEŠKA IR JOS MODIFIKACIJOS KOMIVOJAŽIERIAUS UŽDAVINIUI

Ieva Eimontienė, Alfonsas Misevičius

Kauno technologijos universitetas, Multimedijos inžinertijos katedra,
Studentų g. 50–400a/416a, LT–51368 Kaunas
Tel. 8 686 47340, 8 37 300372
El. paštas: ieva.a@centras.lt, alfonsas.misevicius@ktu.lt

Santrauka

Vieni iš plačiai taikomų euristinių algoritmų kombinatorinio optimizavimo (KO) uždaviniams spręsti yra tabu paieškos (TP) principu pagrįsti algoritmai. Labai perspektyvus TP patobulinimas — vadinamoji iteratyvioji tabu paieška (ITP). Šiame straipsnyje nagrinėjamas iteratyviosios tabu paieškos metodo taikymas, sprendžiant žinomą KO uždavinį — komivojažieriaus uždavinį (angl. *traveling salesman problem*) (KU). Pasiūlyti keli ITP algoritmo variantai, besiskiriantys panaudojamais sprendinių diversifikavimo (mutavimo) operatoriais. Šie variantai išbandyti su testiniais KU pavyzdžiais iš testinių pavyzdžių bibliotekos TSPLIB. Gauti eksperimentų rezultatai rodo, jog naujas pasiūlytas inversijos ir įterpimo mutavimo operatorius padidina ITP efektyvumą ir yra pranašesnis už kitus diversifikavimo operatorius didelei daliai testinių KU pavyzdžių.

1. Įvadas

Komivojažieriaus uždavinys (KU) [4] formuluojamas taip: duota atstumų tarp tam tikrų objektų (juos priimta vadinti miestais) matrica $\boldsymbol{D} = (d_{ij})_{n\times n}$ bei aibė Π, kurią sudaro visi galimi natūrinių skaičių nuo 1 iki n perstatymai. Reikia surasti tokį perstatymą $p \in \Pi$, kuriam esant būtų minimizuota tikslo funkcija:

$$z(p) = \sum_{i=1}^{n-1} d_{p(i)p(i+1)} + d_{p(n)p(1)}. \qquad (1)$$

Perstatymui (KU sprendiniui) $p = (p(1), p(2), ..., p(n))$ $(p(i) \in \{1, 2, ..., n\})$ atitinka n miestų seka, t.y. maršrutas. Perstatymo elementas $p(i)$ žymi i-tąjį aplankytą maršruto miestą, tuo tarpu perstatymo poros $(p(1), p(2))$, ..., $(p(i), p(i+1))$, ..., $(p(n), p(1))$ yra traktuojamos kaip maršruto atkarpos, o $z(p)$ reiškia maršruto ilgį. Tokiu būdu KU tikslą galima suformuluoti ir taip: surasti trumpiausią galimą uždarą maršrutą, kuriame kiekvienas miestas aplankomas tik vieną kartą.

Komivojažieriaus uždavinys priklauso NP-sunkių kombinatorinio optimizavimo uždavinių klasei ir yra savotiška eksperimentavimo bazė, kuriant ir tiriant įvairius algoritmus [1,3,4,6,8]. Toliau nagrinėjamas euristinis iteratyviosios tabu paieškos (ITP) metodas būtent šiam uždaviniui.

Straipsnio struktūra yra tokia. Iš pradžių apžvelgiamas iteratyviosios tabu paieškos principas. 3 sk. aprašomos ITP algoritmo modifikacijos komivojažieriaus uždaviniui. Eksperimentinių tyrimų rezultatai, gauti su įvairiais testiniais KU pavyzdžiais, pateikiami 4 sk. Straipsnis baigiamas išvadomis.

2. Iteratyvioji tabu paieška

Iteratyvioji tabu paieška (ITP) yra savarankiškas bendresnio euristinio metodo — iteratyviosios lokaliosios paieškos (ILP) [7] — atvejis. Esminė ILP idėja slypi „griauti ir atstatyti“ metodika pagrįstos paieškos strategijos taikymu nagrinėjant sprendinių erdvę. Iteraciniu būdu gerinant sprendinius (intensyvinant paiešką) bei juos kaskart pertvarkant, transformuojant (diversifikuojant paiešką) ir taip pereinant nuo vieno lokaliai optimalaus sprendinio prie kito, stengiamasi aprėpti kiek įmanoma didesnius sprendinių poerdvius, taip sudarant palankesnes sąlygas artėti prie globaliojo optimumo. Iteratyvioji tabu paieška charakterizuojama šiais pagrindiniais komponentais: lokaliąja paieška (sprendinio lokaliojo pagerinimo (SLP) procedūra), diversifikavimu (sprendinio pertvarkymo — toliau vartosime terminą „mutavimo“ — operatoriumi) ir sprendinio-kandidato atrinkimo kriterijumi.

Iteratyvioji lokalioji paieška pradedama duoto sprendinio (KU atveju tai yra kuris nors perstatymas p) pagerinimu, gaunant lokaliai optimalų sprendinį, tarkime $p^{\bullet}$. Gautasis

sprendinys yra tam tikru mastu pertvarkomas, gaunant naują sprendinį, pvz. $p^{\sim}$. Rekonstruotas sprendinys $p^{\sim}$ vaidina iš naujo startuojančios lokaliojo pagerinimo procedūros pradinio sprendinio vaidmenį. SLP procedūra grąžina naują lokaliai optimalų sprendinį $p^{\bullet}$, šis vėl rekonstruojamas ir t.t.

ITP metodo struktūra yra labai panaši [9]. Esminis ITP skiriamasis bruožas yra tas, kad lokaliosios paieškos vaidmenyje yra tabu paieškos (TP) [2] algoritmas. Priminsime, kad pagrindinė standartinės TP idėja yra leidimas atlikti sprendinių perėjimus net ir tais atvejais, kai naujasis sprendinys iš duotojo sprendinio aplinkos[‡] nėra geresnis už esamą sprendinį. Tam tikri perėjimai uždraudžiami tam, kad būtų neleidžiama grįžti į tuos pačius, jau nagrinėtus sprendinius. Kartu su sprendinių draudimais naudojamas ir vadinamasis aspiracijos kriterijus [2].

3. Iteratyvioji tabu paieška ir jos modifikacijos komivojažieriaus uždaviniui

Sudaryto ITP algoritmo komivojažieriaus uždaviniui veikimas pagrįstas schemos $(Q, \tau, 1)$ taikymu. Remiantis šia schema, bendras globalių (t.y. iteratyviosios tabu paieškos) iteracijų skaičius yra lygus Q. Kiekvienos iš Q iteracijų metu atliekama τ lokalių (t.y. sprendinio lokaliojo pagerinimo, panaudojant tabu paiešką) iteracijų ir viena sprendinio mutavimo iteracija (vienas kreipimasis į mutavimo procedūrą). Šiuo atveju sandauga $Q\tau$ reiškia bendrą iteracijų skaičių, o santykis Q/τ gali būti traktuojamas kaip diversifikavimo (mutacijų) dažnumas.

ITP metodo veikimo schema, panaudojant formalią aprašymo pseudo-kalbą, pateikta 1 pav.

```
procedure IteratyviojiTabuPaieška;
// pradiniai duomenys: p – esamas sprendinys, Q – bendras (globalių) ITP iteracijų skaičius,
//                     τ – vidinių (lokalių) TP iteracijų skaičius, μ – mutavimo lygis (stiprumas)
// rezultatai: p* – geriausias surastas sprendinys
begin
  optimizuoti sprendinį p atliekant τ (standartinės) tabu paieškos iteracijų,
  tegul optimizuotas sprendinys yra p•;
  p* := p•;
  for q := 1 to Q do begin // pagrindinis iteratyviosios tabu paieškos ciklas
    parinkti sprendinį-kandidatą p mutavimui;
    atlikti sprendinio p mutavimą, kai mutavimo lygis μ,
    tegul mutuotas sprendinys yra p~;
    pagerinti sprendinį p~ atliekant τ tabu paieškos iteracijų,
    tegul naujas pagerintas sprendinys yra p•;
    if z(p•)<z(p*) then p* := p• // rastas geresnis (lokaliai optimalus) sprendinys įsimenamas
  end // for
end.
```

1 pav. Iteratyviosios tabu paieškos metodo aprašas, panaudojant pseudo-kalbą

3.1. Sprendinių lokaliojo pagerinimo procedūra

Pradinis ITP algoritmo sprendinys generuojamas atsitiktiniu būdu. Mes remiamės prielaida, jog galutinių rezultatų gerumą lemia ne pradinis sprendinys, bet vėlesnis iteracinis sprendinių gerinimo procesas.

Sprendinių lokaliajam gerinimui naudojamas [8] straipsnyje aprašytas tabu paieškos algoritmas. Labai trumpai šio algoritmo funkcionavimas galėtų būti nusakytas taip. TP procesas pradedamas nuo pradinio

[‡] Sprendinio p aplinka vadinamas poaibis $\Pi' \subseteq \Pi$, sudarytas iš sprendinių, „pakankamai artimų“ sprendiniui p. Aplinka gaunama, panaudojant tam tikrą aplinkos funkciją Θ: $\Pi \to 2^{\Pi}$ (2^{Π} žymi aibės Π visų galimų poaibių aibę).

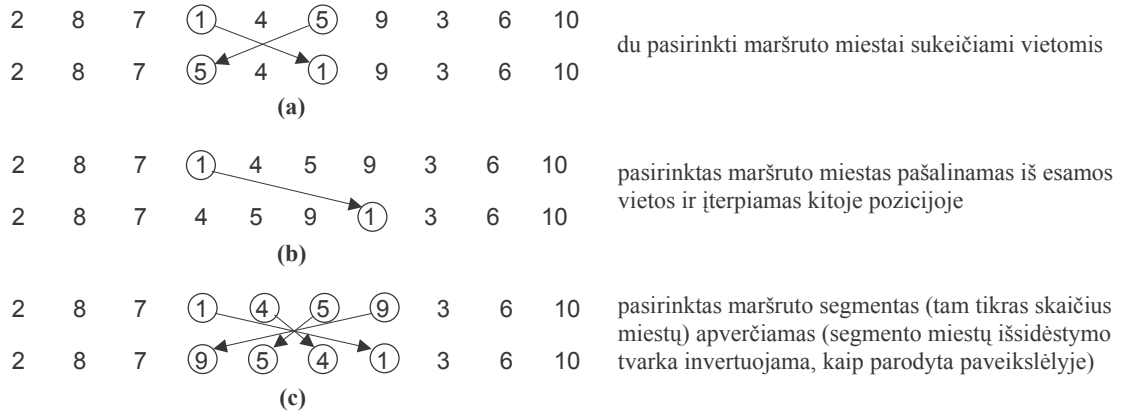
sprendinio, t.y. maršruto p, po to perėjimai iš vieno sprendinio į kitą vykdomi iteraciniu būdu. Paieškos eigoje analizuojama einamojo sprendinio p aplinka $\Theta(p)$ ir pereinama į tą sprendinį $p' \in \Theta(p)$, kuriam tikslo funkcijos z reikšmė yra mažiausia (perėjimas gali būti atliekamas ir tuo atveju, kai tai pablogina tikslo funkcijos reikšmę (!)). Grįžimas į ankstesnius sprendinius turi būti uždraustas tam tikram laikotarpiui — kad būtų išvengta paieškos kartojimo nuo to paties „taško“. Tai realizuojama, įtraukiant nagrinėtus sprendinius į specialų sąrašą (atmintį) T. Tokiu būdu perėjimas į sprendinį $p' \in \Theta(p)$ yra tabu, jeigu tas sprendinys duotu momentu yra sąraše T.

Išsamus TP algoritmo KU aprašymas yra duotas [8] straipsnyje.

3.2. Sprendinių mutavimas

Sprendinių mutavimo paskirtis — diversifikuoti paieškos procesą, „išsklaidant“ paiešką kiek įmanoma įvairesnėse sprendinių erdvės dalyse. Mutavimo metu nėra tikslo duotojo sprendinio visiškai suardyti, sugriauti; greičiau atvirkščiai, generuojant naują sprendinį, siekiama, jog gautas sprendinys paveldėtų tam tikras geras ankstesnio sprendinio charakteristikas, bet tuo pačiu turi būti užtikrinamas ir pakankamas atstumas tarp esamo ir naujai gauto sprendinio.

Sprendžiant komivojažieriaus uždavinį, įmanomi įvairūs sprendinių mutavimo būdai [5]. Galimų mutavimo operatorių pavyzdžiai yra: sukeitimo mutacija, įterpimo mutacija, inversijos (apvertimo) mutacija ir kt. Minėtų mutavimo operatorių iliustracijos pateiktos 2 pav.



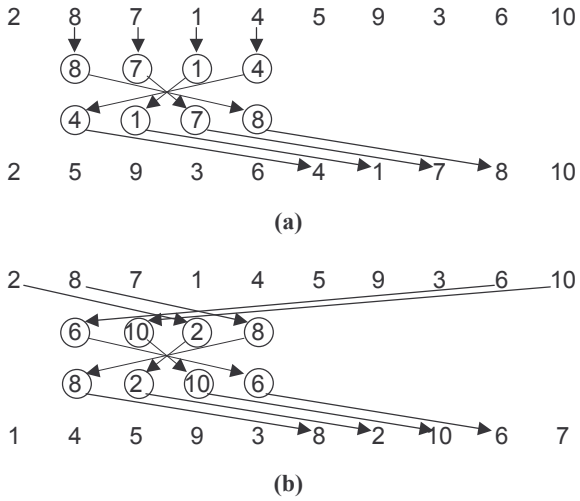
2 pav. Mutavimo operatorių iliustravimas: sukeitimo mutacija (a), įterpimo mutacija (b), inversijos (apvertimo) mutacija (c)

Siekiant padidinti ITP algoritmo efektyvumo laipsnį, pasiūlyti patobulinti mutavimo operatoriai — tai inversijos ir įterpimo mutavimo operatorius bei išplėstinis inversijos ir įterpimo mutavimo operatorius.

3.2.1 Inversijos ir įterpimo mutavimo operatorius

Inversijos ir įterpimo mutavimo (IĮM) operatoriuje apjungtas sprendinio elementų invertavimas (apvertimas) ir invertuotų elementų įterpimas kitoje sprendinio pozicijoje. Be to, IĮM procedūroje numatytas papildomas inversijos ir įterpimo atvejis, kuris gaunamas, kai invertuojamą segmentą, t.y. sprendinio elementų (maršruto miestų) rinkinį sudaro negretimi sprendinio elementai. Invertuojamų sprendinio elementų skaičius (segmento ilgis) (μ) apibrėžia mutavimo lygį (stiprumą). Tinkamo mutavimo lygio nustatymas yra labai svarbus, siekiant gauti aukštos kokybės sprendinius. Tai padaryti galima eksperimentų būdu (žr. 4 sk.).

IĮM operatoriaus veikimo principas iliustruojamas 3 pav. Formalus inversijos ir įterpimo mutavimo procedūros aprašymas pateiktas 4 pav.



pasirinktas maršruto segmentas pašalinamas iš esamos vietos, apverčiamas (kaip ir inversijos mutacijos atveju), po to įterpiamas kitoje vietoje (panašiai, kaip įterpimo mutacijos atveju)

atskiras inversijos ir įterpimo mutacijos atvejis gaunamas, kai invertuojamą segmentą sudaro negretimi perstatymo elementai

3 pav. Inversijos ir įterpimo mutavimo operatoriaus iliustravimas: standartinis inversijos atvejis (a), specialus inversijos atvejis (b)

3.2.2 Išplėstinio inversijos ir įterpimo mutavimo operatorius

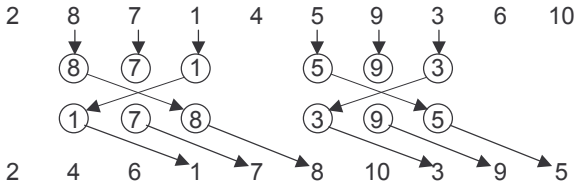
Bendresnis inversijos ir įterpimo mutavimo atvejis yra patobulinto, išplėstinio inversijos ir įterpimo mutavimo (IIĮM) operatorius. Patobulinimo esmė — invertavimą ir įterpimą taikyti ne vienam, o keliems sprendinio, t.y. maršruto segmentams. Šiuo atveju yra svarbu deramo segmentų skaičiaus (N_s) parinkimas.

IIĮM operatoriaus funkcionavimo schema iliustruojama 5 pav.

Parenkant sprendinį-kandidatą mutavimui, taikoma vadinamosios „koncentruotos paieškos“ strategija. Šiuo atveju kandidatas mutavimo procedūrai yra geriausias duotu momentu surastas lokaliai optimalus sprendinys. Tai leidžia fokusuoti, suintensyvinti paiešką tam tikrose sprendinių erdvės srityse, būtent elitinių lokaliai optimalių sprendinių aplinkose.

```
procedure InversijosIterpimoMutavimas;
// pradiniai duomenys: p – esamas sprendinys (maršrutas), n – miestų skaičius, μ – mutavimo lygis (stiprumas)
// rezultatai: p~ – mutuotas sprendinys (maršrutas)
begin
  tegul u, v (u≠v) – atsitiktiniai skaičiai iš intervalo [1, n],
 čia u yra pradinės invertuojamo segmento pozicijos maršrute p numeris,
 v – įterpimo pozicijos numeris;
  if u + μ – 1 ≤ n then kopijuoti(p, u, s, 1, μ)
                   else begin kopijuoti(p, u, s, 1, n – u + 1); kopijuoti(p, 1, s, n – u + 2, μ – n + u + 1);
end;
  s̄ := inversija(s) ;  // segmento s elementai invertuojami, gaunant segmentą s̄
  if v + μ – 1 ≤ n then kopijuoti( s̄ , 1, p~ , v, μ)
                   else begin kopijuoti( s̄ , 1, p~ , v, n – v + 1);
                              kopijuoti( s̄ , n – v + 2, p~ , 1, μ – n + v + 1) end;
  perkelti likusius elementus iš p į p~
end.
```

4 pav. Inversijos ir įterpimo mutavimo procedūros aprašymas.
Pastaba: procedūra kopijuoti(*x*, *i*, *y*, *j*, *k*) **perkelia *k* elementų iš maršruto *x*, pradedant *i*-tąja pozicija, į maršrutą *y*, pradedant *j*-tąja pozicija**



išplėstinės inversijos ir įterpimo mutacijos atvejis gaunamas, kai invertuojama ir įterpiama daugiau negu vienas segmentas

5 pav. Išplėstinio inversijos ir įterpimo mutavimo operatoriaus iliustravimas

1 lentelė. Algoritmų palyginimo rezultatai

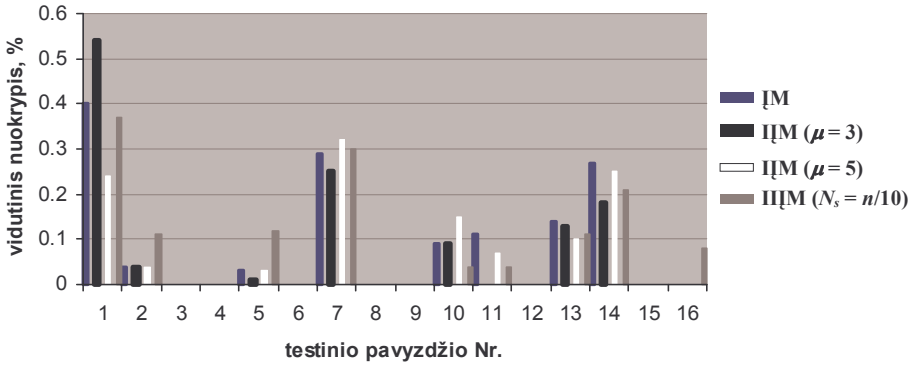
Nr.	Testinis pavyzdys	n	z_{opt}	$\bar{\delta}$, $C_{1\%}/C_{opt}$							
				ĮM		IĮM ($\mu = 3$)		IĮM ($\mu = 5$)		IIĮM ($N_s = n/10$)	
1.	a280	280	2579	0,4	8/0	0,54	8/2	0,24	9/3	0,37	9/1
2.	att48	48	10628	**0,04**	**10/8**	**0,04**	**10/8**	**0,04**	**10/8**	0,11	10/6
3.	bayg29	29	1610	**0**	**10/10**	**0**	**10/10**	**0**	**10/10**	**0**	**10/10**
4.	berlin52	52	7542	**0**	**10/10**	**0**	**10/10**	0	**10/10**	**0**	**10/10**
5.	bier127	127	118282	0,03	**10/9**	**0,01**	**10/9**	0,03	10/8	0,12	10/6
6.	burma14	14	3323	**0**	**10/10**	**0**	**10/10**	**0**	**10/10**	**0**	**10/10**
7.	eil101	101	6294	0,29	**10/5**	**0,25**	10/3	0,32	9/5	0,3	9/5
8.	gr17	17	2085	**0**	**10/10**	**0**	**10/10**	**0**	**10/10**	**0**	**10/10**
9.	gr48	48	5046	**0**	**10/10**	**0**	**10/10**	**0**	**10/10**	**0**	**10/10**
10.	gr96	96	55209	0,09	10/4	0,09	10/5	0,15	10/4	**0,04**	**10/7**
11.	kroa100	100	21282	0,11	10/6	**0**	**10/10**	0,07	10/7	0,04	10/6
12.	pr76	76	108159	**0**	**10/10**	**0**	**10/10**	**0**	**10/10**	**0**	**10/10**
13.	pr107	107	44303	0,14	10/4	0,13	10/6	**0,1**	**10/7**	0,11	10/5
14.	pr299	299	48191	0,27	10/0	**0,18**	**10/2**	0,25	10/0	0,21	10/1
15.	st70	70	675	**0**	**10/10**	**0**	**10/10**	**0**	**10/10**	**0**	**10/10**
16.	u159	159	42080	**0**	**10/10**	**0**	**10/10**	**0**	**10/10**	0,08	10/9

4. Eksperimentų rezultatai

Siekiant ištirti sudarytų ITP modifikacijų efektyvumą, buvo atlikti eksperimentiniai tyrimai su įvairiais testiniais KU pavyzdžiais (duomenimis) iš testinių pavyzdžių bibliotekos TSPLIB [10]. Eksperimentų metu lygintos atskiros ITP modifikacijos, besiskiriančios panaudotomis mutavimo procedūromis. Konkrečiai eksperimentuota su šiais mutavimo operatoriais: a) įterpimo mutacija (ĮM); b) inversijos ir įterpimo mutacija (IĮM), kai mutavimo stiprumas μ lygus 3 (kai $\mu = 1$, IĮM sutampa su ĮM); c) inversijos ir įterpimo mutacija, kai $\mu = 5$; d) išplėstinė inversijos ir įterpimo mutacija (IIĮM), kai segmentų skaičius N_s lygus $n/10$ (n yra miestų skaičius).

Buvo pasirinkti tokie algoritmų efektyvumo įvertinimo kriterijai: a) vidutinis santykinis nuokrypis nuo optimalaus sprendinio — $\bar{\delta}$ ($\bar{\delta} = 100(\bar{z} - z_{opt})/z_{opt}$ [%], čia $\bar{z}$ yra gautų tikslo funkcijos reikšmių (maršrutų ilgių) vidurkis, apskaičiuotas atlikus 10 algoritmo pakartotinių vykdymų, o z_{opt} yra optimali tikslo funkcijos reikšmė (optimalias reikšmes galima sužinoti iš [10])); b) sprendinių, esančių „1% optimalumo intervale“ ($\bar{\delta} \le 1$), skaičius (kai atlikta 10 pakartotinių vykdymų) — $C_{1\%}$; c) rastų optimalių sprendinių skaičius — C_{opt}.

Atskirų ITP modifikacijų palyginimo rezultatai pateikti 1 lentelėje.



6 pav. Vidutinio santykinio nuokrypio reikšmių pasiskirstymas pagal skirtingus testinius pavyzdžius

Matyti, jog algoritmų rezultatai priklauso nuo testinių duomenų tipo (žr. 6 pav.). Tačiau didelei daliai testinių KU pavyzdžių inversijos ir įterpimo mutavimo operatorius, atrodo, yra pranašesnis už kitus. Rezultatai, gauti tirtiems testiniams pavyzdžiams, liudija, jog mutavimo lygis (segmento ilgis), naudojant IĮM operatorių, taip pat įtakoja sprendinių kokybę. Geriausius rezultatus mums pavyko gauti, kai segmento ilgis lygus 3. Vis tik, norint pasiekti objektyvesnius rezultatus, reikėtų atlikti papildomus eksperimentus.

5. Išvados

Pastaruoju metu kombinatorinio optimizavimo (KO) uždaviniams spręsti plačiai taikomi tabu paieškos metodika pagrįsti algoritmai. Viena perspektyvių TP metodo versijų yra iteratyvioji tabu paieška (ITP). Šiame straipsnyje pateikiami rezultatai, gauti išbandžius iteratyviąją tabu paiešką sudėtingam gerai žinomam KO uždaviniui — komivojažieriaus uždaviniui.

Pasiūlyti keli iteratyviosios tabu paieškos variantai, besiskiriantys panaudojamais sprendinių diversifikavimo, t.y. mutavimo operatoriais. Šie variantai išmėginti su testiniais KU pavyzdžiais iš testinių pavyzdžių bibliotekos TSPLIB.

Gauti eksperimentų rezultatai rodo, jog naujas pasiūlytas inversijos ir įterpimo mutavimo (IĮM) operatorius padidina ITP efektyvumą, leidžia per priimtiną vykdymo laiką gauti sprendinius, kurie yra optimalūs arba labai artimi optimaliems. Rezultatai taip pat liudija, kad IĮM operatorius, parinkus tinkamą mutavimo lygį, yra pranašesnis už kitus mutavimo variantus didelei daliai nagrinėtų testinių KU pavyzdžių.

Sudarytos IĮM ir IIĮM procedūros galėtų būti toliau modifikuojamos ir tobulinamos.

Literatūros sąrašas

[1] **J.L.Bentley.** Experiments on traveling salesman heuristics. Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco,USA), 1990, 91-99.

[2] **F.Glover, M.Laguna.** *Tabu Search*, Dordrecht: Kluwer, 1997.

[3] **D.S.Johnson.** Local optimization and the traveling salesman problem. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science*, vol.443, Berlin: Springer, 1990, 446-461.

[4] **D.S.Johnson, L.A.McGeoch.** The traveling salesman problem: a case study. In E.Aarts, J.K.Lenstra (eds.), *Local Search in Combinatorial Optimization*, Chichester: Wiley, 1997, 215-310.

[5] **P.Larranaga, C.M.H.Kuijpers, R.H.Murga, I.Inza, S.Dizdarevic.** Genetic algorithms for the traveling salesman problem: a review of representations and operators. *Artificial Intelligence Review*, 1999, vol.13, 129-170.

[6] **S.Lin, B.W.Kernighan.** An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 1973, vol.21, 498-516.

[7] **H.R.Lourenco, O.Martin, T.Stützle.** Iterated local search. In F.Glover, G.Kochenberger (eds.), *Handbook of Metaheuristics*, Norwell: Kluwer, 2002, 321-353.

[8] **A.Misevičius**. Iteratyviosios tabu paieškos algoritmas komivojažieriaus uždaviniui. *Informacinės technologijos 2005: konferencijos «Lietuvos mokslas ir pramonė» pranešimų medžiaga*, Kaunas, 2005, 91-101.

[9] **A.Misevičius, A.Lenkevičius, D.Rubliauskas.** Iterated tabu search: an improvement to standard tabu search. *Information Technology and Control*, 2006, vol.35, no.3, 187-197.

[10] **G.Reinelt.** TSPLIB – A traveling salesman problem library. *ORSA Journal on Computing*, 1991, vol.3-4, 376-385. [Žr. taip pat `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`.]

ITERATED TABU SEARCH AND ITS MODIFICATIONS FOR THE TRAVELING SALESMAN PROBLEM

Summary

Tabu search (TS) algorithms are among those widely used for combinatorial optimization problems. One of the promising enhancements of tabu search is the iterated tabu search (ITS). In this paper, we discuss the iterated tabu search approach for the famous optimization problem, the traveling salesman problem (TSP). In particular, several modifications of the ITS algorithm are proposed. These modifications are examined on a set of the TSP test instances taken from the TSP library TSPLIB. The results obtained from the experiments show that the new proposed ITS modification based on an inversion and insertion mutation operator is superior to other ITS variants on a large part of the TSP instances examined.