Vilniaus universiteto leidykla

Contents lists available at Vilnius University Press

Informacijos mokslai
2019, vol. 85

ISSN 1392-0561 eISSN 1392-1487
DOI: https://doi.org/10.15388/Im.2019.85.19

Dviejų lygių iteracinis tabu paieškos algoritmas kvadratinio paskirstymo uždaviniui

Alfonsas Misevičius

Kauno technologijos universiteto
Multimedijos inžinerijos katedros tech. m. dr., profesorius
Kaunas University of Technology,
Department of Multimedia Engineering, Dr., Prof.
Studentų g. 50-400/416a, LT-51368 Kaunas
El. paštas alfonsas.misevicius@ktu.lt

Dovilė Kuznecovaitė (Verenė)

Kauno technologijos universiteto
Multimedijos inžinerijos katedros doktorantė
Kaunas University of Technology,
Department of Multimedia Engineering, PhD Student
Studentų g. 50-408, LT-51368 Kaunas
El. paštas dovile.kuznecovaite@ktu.lt

Santrauka. Šiame straipsnyje nagrinėjamas vadinamasis dviejų lygių iteracinis tabu paieškos (ITP) algoritmas kvadratinio paskirstymo (KP) uždaviniui. Algoritmo naujumas yra tas, jog į jį yra integruotos sprendinių mutavimo procedūros, kurių esminė paskirtis yra diversifikuoti paieškos procesą, išvengiant paieškos stagnacijos ir taip padidinant jos efektyvumą. Algoritmo veikimas išbandytas su įvairių tipų mutavimo procedūrų realizavimo variantais. Atlikti kompiuteriniai eksperimentai su KP uždavinio testavimo duomenų pavyzdžiais iš standartinių pavyzdžių bibliotekos QAPLIB. Pateikti eksperimentų rezultatai, kurie iliustruoja, kaip skirtingos prigimties mutavimo procedūros, esančios ITP sudėtyje, gali įvairiai paveikti algoritmo efektyvumą.
Pagrindiniai žodžiai: skaitmeninis intelektas, kombinatorinis optimizavimas, euristiniai optimizavimo algoritmai, tabu paieška, mutavimo procedūros, kvadratinio paskirstymo uždavinys.

A 2-Level Iterated Tabu Search Algorithm for the Quadratic Assignment Problem

Summary. In this paper, a 2-level iterated tabu search (ITS) algorithm for the solution of the quadratic assignment problem (QAP) is considered. The novelty of the proposed ITS algorithm is that the solution mutation procedures are incorporated within the algorithm, which enable to diversify the search process and eliminate the search stagnation, thus increasing the algorithm's efficiency. In the computational experiments, the algorithm is examined with various implemented variants of the mutation procedures using the QAP test (sample) instances from the library of the QAP instances – QAPLIB. The results of these experiments demonstrate how the different mutation procedures affect and possibly improve the overall performance of the ITS algorithm.
Keywords: computational intelligence, combinatorial optimization, heuristic algorithms, tabu search, mutation procedures, quadratic assignment problem.

Received: 03/05/2019. **Accepted:** 04/09/2019

Įvadas

Kvadratinio paskirstymo (KP) uždavinys (angl. *quadratic assignment problem* (*QAP*)) matematiškai formuluojamas taip. Duotos dvi neneigiamų sveikųjų skaičių matricos $\boldsymbol{A} = (a_{ij})_{n\times n}$ ir $\boldsymbol{B} = (b_{kl})_{n\times n}$, taip pat aibė Π_n, kurią sudaro visi galimi natūrinių skaičių nuo 1 iki n perstatymai[1]. Tikslas yra surasti perstatymą $p = (p(1), p(2), \dots, p(n)) \in \Pi_n$ ir tokį, jog būtų minimizuota ši funkcija:

$$p = (p(1), p(2), \dots, p(n)) \in \Pi_n . \quad (1)$$

Dėstymo teorijos (angl. *location theory*) kontekste su KP uždaviniu gali būti modeliuojamas n įrengimų išdėstymas į n paskyrimo vietų, turint tikslą minimizuoti bendrą kainą, atsižvelgiant į įrengimų susietumą ir atstumus tarp paskyrimo vietų (Koopmans, Beckmann, 1957). Čia matricos $\boldsymbol{A}$ reikšmės asocijuojamos su įrengimų susietumo svoriais, o matricos $\boldsymbol{B}$ reikšmės nurodo atstumus tarp paskyrimo vietų.

Dar vienas kvadratinio paskirstymo uždavinio taikymo pavyzdys – tai elektroninių komponentų išdėstymas spausdinto montažo plokštės (ar kristalo) pozicijose (Hanan, Kurtzberg, 1972). Šiame kontekste matricos $\boldsymbol{A}$ elementai yra elektrinių sujungimų tarp komponentų porų kiekiai. Matricos $\boldsymbol{B}$ elementai atitinka atstumus tarp fiksuotų pozicijų montažinėje plokštėje (kristale). Tuomet perstatymas p = (p(1), p(2), …, $p(n)$) gali būti interpretuojamas kaip atskira komponentų išdėstymo pozicijose konfigūracija. Perstatymo elementas $p = (p(1), p(2), \dots, p(n)) \in \Pi_n$ šiuo atveju nurodo numerį pozicijos, į kurią paskirtas i-tasis komponentas. Tokiu būdu z (tiksliau, $z/2$) gali būti suprantamas kaip bendras sujungimų tarp komponentų ilgis, kuomet visi n komponentai yra išdėstyti atitinkamose n pozicijų. (Literatūroje galima rasti ir dar daugiau KP uždavinio praktinio taikymo pavyzdžių aprašymų (Çela, 1998).)

Kita vertus, KP uždavinys yra matematinis uždavinys, t. y. kombinatorinio optimizavimo uždavinys. Šiuo atveju tikslas yra surasti optimalų sprendinį, kur sprendiniai yra išreiškiami natūrinių skaičių perstatymais. Optimalumo kriterijus aprašomas (1) formule, o funkcija z yra vadinama tikslo funkcija (TF).

Įrodyta, jog KP uždavinys priklauso NP sunkių optimizavimo uždavinių klasei (Sahni, Gonzalez, 1976). Tai reiškia, kad skaičiavimų laikas, reikalingas optimumui pasiekti, yra susietas su uždavinio apimtimi n eksponentine priklausomybe. Dėl šios priežasties KP uždavinio sprendimui naudojami euristiniai optimizavimo algoritmai (Michalewicz, Fogel, 2000; Yang, 2010; Edelkamp, Schrödl, 2012; Siarry, 2016). Nors šie algoritmai negarantuoja optimalaus sprendinio suradimo, tačiau jie leidžia gauti pakankamai aukštos kokybės (artimus optimaliems) sprendinius per priimtiną skaičiavimų laiką. KP uždaviniui sėkmingai išbandyti šie euristiniai algoritmai: atkaitinimo modeliavimas (angl. *simulated annealing*) (Misevičius, 2003), genetiniai algoritmai (angl. *genetic algorithms*) (Merz, Freisleben, 2000; Drezner, 2003; Misevicius, 2004; Tang ir kt., 2006; Ahmed, 2015; Benlic, Hao, 2015; Chmiel,

[1] Priminsime, jog perstatymą p formaliai galima nusakyti rinkiniu $p = (p(1), p(2), \dots, p(n))$; $p(i) \in \{1, 2, \dots, n\}$; $i = 1, 2, \dots, n$; $p(i) \neq p(j)$, kai $i \neq j$; $i, j = 1,2, \dots, n$.

Kwiecień, 2018), tabu paieška (angl. *tabu search*) (Taillard, 1991; Misevicius, 2005; Shylo, 2017), godžiosios randomizuotos adaptyvios paieškos procedūros (angl. *greedy randomized adaptive search procedures* (*GRASP*)) (Li ir kt., 1994), iteratyvioji lokalioji paieška (angl. *iterated local search*) (Stützle, 2006), skruzdėlių kolonijų (Gambardella ir kt., 1999), bičių spiečių (Dokeroglu ir kt., 2019) imitavimo algoritmai (angl. *artificial bee colony*, *ant colony optimization*) ir kt. (Benlic, Hao, 2013; Hafiz, Abdennour, 2016; Aksan ir kt., 2017; Abdel-Basset ir kt., 2018; Chmiel, 2019). Nepaisant pasiektų daug žadančių rezultatų, euristinių algoritmų KP uždaviniui tolesnio išvystymo ir tobulinimo klausimai vis dar išlieka aktuali mokslinių tyrinėjimų tema.

Šiame straipsnyje aprašomas patobulintas tabu paieškos (TP) algoritmas – vadinamasis dviejų lygių iteracinis tabu paieškos (ITP) algoritmas. Algoritmo veikimo principo naujoviškumas yra tas, kad tabu paieškos procedūra yra vykdoma iteraciniu būdu, pradedant vis nuo naujo sprendinio (pereinant tarp skirtingų lokaliai optimalių sprendinių). Iteracinį tabu paieškos algoritmą sudaro šie pagrindiniai komponentai: tabu paieškos procedūra, sprendinio-kandidato parinkimas, parinkto sprendinio mutavimas. TP procedūra atliekama kuo greičiau, panaudojant nedidelį iteracijų skaičių. Gautas TP procedūros sprendinys po to yra perturbuojamas mutavimo procedūroje tam, kad būtų diversifikuotas paieškos procesas, taip bandant išvengti paieškos stagnacijos, kas yra viena pagrindinių kliūčių vykdant standartinę, neiteracinę TP. Siekiant nustatyti galimą efektyviausią ITP modifikaciją, buvo tirta daugiau nei 10 mutavimo proceso realizavimo variantų.

Straipsnio struktūra yra tokia. Iš pradžių pateikiamos bazinės formuluotės. Po to aprašomas dviejų lygių iteracinis tabu paieškos algoritmas ir į jį integruotos sprendinių mutavimo procedūros. Taip pat pateikiami kompiuterinių-eksperimentinių tyrimų rezultatai, gauti išbandžius įvairius mutavimo procedūrų algoritminio realizavimo variantus. Straipsnis baigiamas apibendrinamosiomis pastabomis.

Dviejų lygių iteracinis tabu paieškos algoritmas

Bazinės formuluotės

Pradžioje pateikiame bazines formuluotes (apibrėžimus).

1. Tarkime, kad $p(u)$ ($u = 1, \ldots, n$) ir $p(v)$ ($v = 1, \ldots, n$, $u \neq v$) yra du perstatymo (sprendinio) p elementai, kurie vykdant algoritmą gali būti sukeisti vietomis. Tuomet $p^{u,v}$ apibrėšime taip:

$$p^{u,v}(i) = \begin{cases} p(i), & i \neq u, v \\ p(u), & i = v \\ p(v), & i = u \end{cases}. \qquad (2)$$

Tai reiškia, kad perstatymas $p^{u,v}$ gaunamas iš perstatymo p, tame perstatyme sukeičiant elementus $p(u)$ ir $p(v)$ vietomis. Galima pastebėti, kad galioja tokios lygybės: $p^{u,u} = p$, $p^{u,v} = p^{v,u}$, $(p^{u,v})^{u,v} = p$. Taip pat bet kokiems tarpusavyje skirtingiems u, v, w galioja $(p^{u,v})^{v,w} = (p^{u,w})^{u,v}$, $(p^{u,v})^{u,w} = (p^{v,w})^{u,v}$.

2. Tikslo funkcijos (z) pokytis (Δ), sukeitus sprendinio-perstatymo elementus $p(u)$ ir $p(v)$ vietomis, apskaičiuojamas pagal šią bendrą formulę:

$$\Delta(p^{u,v},p) = z(p^{u,v}) - z(p) = (a_{uu} - a_{vv})\left(b_{p(v)p(v)} - b_{p(u)p(u)}\right) +$$
$$(a_{uv} - a_{vu})\left(b_{p(v)p(u)} - b_{p(u)p(v)}\right) + \quad (3)$$
$$\sum_{k=1,k\neq u,v}^{n}\left[(a_{uk} - a_{vk})\left(b_{p(v)p(k)} - b_{p(u)p(k)}\right) + (a_{ku} - a_{kv})\left(b_{p(k)p(v)} - b_{p(k)p(u)}\right)\right].$$

Tikslo funkcijos pokytį dviem „perstatymams-kaimynams“ p ir p', kai $p' = p^{u,v}$, galima apskaičiuoti ir žymiai greičiau – su sąlyga, jog yra įsimintos (išsaugotos) visos ankstesnių pokyčių reikšmės $\Delta(p^{i,j}, p)$ ($i, j = 1, \ldots, n$). Pokyčiams saugoti pakanka dvimatės matricos, o pokyčio reikšmė gaunama, panaudojant $O(1)$ operacijų (Taillard, 1991). Atlikus elementų $p(u)$ ir $p(v)$ sukeitimą, naujos pokyčių reikšmės $\Delta'(p^{i,j}, p)$ ($i \neq u$, $i \neq v$, $j \neq u$, $j \neq v$) apskaičiuojamos pagal šią formulę:

$$\Delta'\left(p^{i,j},p\right) = \Delta\left(p^{i,j},p\right) +$$
$$\left(a_{iu} - a_{iv} + a_{jv} - a_{ju}\right)\left(b_{p(i)p(u)} - b_{p(i)p(v)} + b_{p(j)p(v)} - b_{p(j)p(u)}\right) + \quad (4)$$
$$\left(a_{ui} - a_{vi} + a_{vj} - a_{uj}\right)\left(b_{p(u)p(i)} - b_{p(v)p(i)} + b_{p(v)p(j)} - b_{p(u)p(j)}\right).$$

Pastaba. Jeigu $i = u$ arba $i = v$, arba $j = u$, arba $j = v$, tai turi būti taikoma (3) formulė. Nors pastarosios formulės sudėtingumas yra $O(n)$, tačiau ji taikoma tik keturioms indeksų poroms, o tai įgalina visas pokyčių reikšmes apskaičiuoti panaudojant tik $O(n^2)$ operacijų. Išimtį sudaro inicializacijos fazė, t. y. pirmoji pokyčių apskaičiavimo iteracija, kurios metu reikia (bet tik vieną kartą) atlikti $O(n^3)$ operacijų.

Jeigu matrica $\boldsymbol{A}$ ir / arba matrica $\boldsymbol{B}$ yra simetrinė(s), tai (3) formulė supaprastėja. Sakykime, kad matrica $\boldsymbol{B}$ yra simetrinė. Tuomet galima transformuoti (asimetrinę) matricą $\boldsymbol{A}$ į simetrinę matricą $\boldsymbol{A'}$; tam pakanka sudėti atitinkamus matricos $\boldsymbol{A}$ elementus. Taip gauname žemiau pateikiamą formulę:

$$\Delta(p^{u,v},p) = \sum_{k=1,k\neq u,v}^{n}(a'_{uk} - a'_{vk})\left(b_{p(v)p(k)} - b_{p(u)p(k)}\right); \quad (5)$$

čia $a'_{uk} = a_{uk} + a_{ku}$, $u = 1, \ldots, n$, $v = 1, \ldots, n$, $u \neq$ v. Analogiškai (4) formulė virsta tokia formule:

$$\Delta'\left(p^{i,j},p\right) = \Delta\left(p^{i,j},p\right) +$$
$$\left(a'_{iu} - a'_{iv} + a'_{jv} - a'_{ju}\right)\left(b_{p(i)p(u)} - b_{p(i)p(v)} + b_{p(j)p(v)} - b_{p(j)p(u)}\right). \quad (6)$$

Jeigu $i = u(v)$ arba $j = u(v)$, tai turi būti taikoma (5) formulė.

Disponuojant pakankamos apimties kompiuterine atmintimi, galima vietoje dvimačių matricų $\boldsymbol{A}$ ir $\boldsymbol{B}$ panaudoti atitinkamas trimates matricas $\boldsymbol{A''} = (a''_{uvk})_{n\times n\times n}$ ir $\boldsymbol{B''} = (\boldsymbol{b''}_{lrt})_{n\times n\times n}$. Tarkime, kad $a''_{uvk} = a'_{uk} - a'_{vk}$, o $b''_{lrt} = b_{lt} - b_{rt}$, čia $l = p(j)$, $r = p(i)$, $t = p(k)$.

Tuomet tikslo funkcijos pokyčius galime apskaičiuoti panaudodami labai kompaktiškas formules:

$$\Delta(p^{u,v}, p) = \sum_{k=1, k\neq u,v}^{n} a''_{uvk} b''_{p(v)k(u)p(k)}, \quad (7)$$

$$\Delta'(p^{i,j}, p) = \Delta(p^{i,j}, p) + (a''_{iju} - a''_{ijv})(b''_{p(j)p(i)p(v)} - b''_{p(j)p(i)p(u)}). \quad (8)$$

Matricoms $\boldsymbol{A''}$ ir $\boldsymbol{B''}$ suformuoti panaudojama $O(n^3)$ operacijų; tas atliekama dar prieš pradedant vykdyti algoritmą. Taigi, tai neturi kiek nors žymesnės įtakos bendram algoritmo sudėtingumui; atvirkščiai, atlikti praktiniai eksperimentai patvirtino išankstinę prielaidą, jog galima sutaupyti iki 20 proc. procesorinio laiko. (Aišku, kompiuterinės atminties sunaudojimas išauga: matricų $\boldsymbol{A''}$, $\boldsymbol{B''}$ saugojimui reikia $O(n^3)$ atminties.)

3. Atstumas (vadinamasis Hemingo atstumas) tarp dviejų perstatymų p_1 ir p_2 apibrėžiamas kaip $\delta(p_1, p_2) = |\{i: p_1(i) \neq p_2(i)\}|$. Galioja tokios lygybės: $\delta(p, p) = 0$, $\delta(p_1, p_2) \neq 1$, $0 \leq \delta(p_1, p_2) \leq n$, $\delta(p_1, p_2) = \delta(p_2, p_1)$, $\delta(p, p^{u,v}) = 2$. Pastebėsime, jog bet kuriems u, v, w ($u \neq v$, $u \neq w$, $v \neq w$) $\delta(p, (p^{u,v})^{v,w}) = \delta(p, (p^{v,w})^{u,v}) = \delta(p, (p^{u,v})^{u,w}) = \delta(p, (p^{u,w})^{u,v}) = \delta(p, (p^{u,w})^{v,w}) = \delta(p, p^{v,w})^{u,v}) = 3$. Apskritai, jei turime k skirtingų $u_1, u_2, \ldots, u_k$, tai yra teisinga ši lygybė: $\delta(((((p^{u_1,u_2})^{u_2,u_3})^{\cdots})^{u_{k-1},u_k})) = k$.

4. Sprendinio-perstatymo p aplinka (porinių sukeitimų aplinka) Θ_2 apibrėžiama pagal formulę: $\Theta_2(p) = \{p' : p' \in \Pi_n, p' \neq p, \delta(p, p') \leq 2\}$. Kitaip tariant, aplinką $\Theta_2(p)$ sudaro sprendiniai: $p^{1,2}, p^{1,3}, \ldots, p^{1,n}, p^{2,3}, \ldots, p^{2,n}, \ldots, p^{i-1,n}, p^{i,i+1}, \ldots, p^{i,n}, \ldots, p^{n-1,n}$. Tokiu būdu aplinkos Θ_2 dydis $|\Theta_2|$ yra lygus $(n(n-1))/2$. Tiek žingsnių reikia atlikti norint visiškai išnagrinėti esamo sprendinio p aplinką $\Theta_2(p)$. Taigi, šiai aplinkai išnagrinėti pakanka $O(n^2)$ operacijų. Gali būti apibrėžiamos ir kitokios aplinkos (bendruoju atveju sprendinio p aplinką žymėsime $\Theta(p)$). Sprendinių aplinkų visuma sudaro sprendinių paieškos aibę (sprendinių paieškos erdvę).

5. Sprendinys $p\bullet \in \Pi_n$ vadinamas lokaliai optimaliu sprendiniu aplinkos Θ atžvilgiu, jeigu kiekvienam sprendiniui p' iš aplinkos $\Theta(p\bullet)$ galioja $z(p\bullet) \leq z(p')$ (kitaip tariant, sprendinio $p\bullet$ aplinkoje $\Theta(p\bullet)$ nėra nė vieno geresnio sprendinio už $p\bullet$).

6. Mutavimo (angl. *mutation*) procesas sprendinių-perstatymų atveju gali būti apibrėžiamas panaudojant formalų operatorių $\varphi: \Pi_n \to \Pi_n$. Taigi, jeigu $p\sim = \varphi(p)$, tai $p\sim \in \Pi_n$; be to, $p\sim \neq p$. Galima naudoti ir labiau formalizuotą operatorių $\varphi(p, \mu)$: $\Pi_n \times \mathrm{N} \to \Pi_n$, kuris esamą sprendinį p transformuoja į naują sprendinį $p\sim$ ir tokį, jog $\delta(p, p\sim) = \delta(p, \varphi(p)) = \mu$. Skaitytojas galėtų įsitikinti, kad bet kuriam mutuotam sprendiniui-perstatymui $p\sim$ egzistuoja skaičių seka $u_1, u_2, \ldots, u_t, \ldots$, ir tokia, jog $p\sim = \varphi(\varphi(\varphi(\varphi(\varphi(p, u_1, u_2), u_2, u_3)\ldots), u_{t-1}, u_t), u_t, u_{t+1})\ldots) = (((((p^{(u_1, u_2)})^{u_2, u_3})^{\cdots})^{u_{t-1}, u_t})^{u_{t,t+1}}) \cdots$. Dydį (parametrą) μ vadinsime mutavimo laipsniu (stiprumu). Akivaizdu, jog $2 \leq \mu \leq n$. Taigi, po mutavimo pakinta $100\mu/n$ procentų sprendinio elementų. Parametras μ yra iš esmės vienintelis kiekybinis mutavimo procesą reguliuojantis veiksnys. Kuo didesnė μ reikšmė, tuo daugiau sprendinio elementų sukeičiama (tuo didesniu mastu pakinta sprendinys, tuo labiau mutuotas sprendinys „nutolsta“ nuo esamo sprendinio) ir atvirkščiai. Galima šiuo atveju manyti, kad mutavimo operatorius $100\mu/n$ transformuoja esamą sprendinį p į kitą sprendinį, priklausantį spren-

dinio p aplinkai $\Theta_{\mu}(p)$. Dar sakoma, jog atliekamas perėjimas (μ perėjimas) iš sprendinio p į sprendinį $p\sim$ (sprendinio p pakeitimas sprendiniu $p\sim$). Atskiras mutavimo operatoriaus atvejis yra dviejų perstatymo elementų sukeitimo operatorius $\varphi(p, u, v)$: $\Pi_n \times N \times N \rightarrow \Pi_n$, kuris sukeičia perstatymo u-tąjį ir v-tąjį elementus vietomis, t. y. $\varphi(p, u, v) = p^{u,v}$. Daugelis euristinių algoritmų KP uždaviniui remiasi būtent šio operatoriaus ir aplinkos Θ_2, kaip nereikalaujančios didelio skaičiavimų kiekio, panaudojimu.

Grįždami prie mutavimo, pastebėsime, jog mutavimas kartu su kryžminimu yra pagrindiniai evoliucinių (genetinių) algoritmų operatoriai. Trumpai sakant, mutavimui savybingas grynai atsitiktinis pobūdis, mutavimo procesas savo prigimtimi yra indeterminuotas procesas. Tuo mutavimas iš esmės skiriasi nuo kryžminimo procedūrų. Mutavimo proceso metu generuojamos, apskritai kalbant, naujos sprendinių savybės, nauja informacija; o kryžminimo procedūros paprastai tik (re)kombinuoja informaciją, esančią pirmtakuose. Skirtingai negu standartinėse kryžminimo procedūrose, kur yra paveldimas sukauptas genetinis kodas iš dviejų pirmtakų ir daugiau niekur kitur, mutavimo procedūrose pagrindinis dalykas yra ne informacijos perdavimas, o kitos, naujos informacijos įnešimas, informacijos, kurios prieš tai dar nebuvo, sukūrimas. Šiuo požiūriu mutavimo procesas yra labiau kaip potencialaus naujumo generavimo procesas, kaip labiau kuriantysis, užuomazginis procesas, palyginti su kryžminimu. Be to, mutavimui nebūtina sprendinių populiacija, pakanka tik vieno sprendinio.

Tabu paieškos algoritmas

Skirtingai nuo tradicinių lokaliosios paieškos algoritmų, kurie apsiriboja vieno lokaliai optimalaus sprendinio suradimu, TP algoritmai tęsia paiešką ir tuo atveju, kai gauto sprendinio aplinkoje nebeegzistuoja geresnių sprendinių-kaimynų. Esminė TP idėja yra ta, jog yra (laikinai) uždraudžiama grįžti į tuos pačius, neseniai nagrinėtus sprendinius tam, kad būtų išvengta paieškos ciklinimosi, t. y. kartojimo nuo to paties išeities „taško“ (Glover, Laguna, 1997). Tam tikrais periodais atskiri sprendiniai yra uždraudžiami („įšaldomi“), skelbiami „tabu“. Laikinų draudimų strategija leidžia „neįstrigti“ lokaliųjų optimumų zonose ir suteikia galimybes po laikinų TF pablogėjimų tęsti paiešką naujose sprendinių erdvės srityse. Taip galima pereiti nuo vieno lokaliai optimalaus sprendinio prie kito, tikėtina, geresnio; tai kartojant daug kartų, žymiai išauga tikimybė surasti aukštesnės kokybės sprendinius.

Vykstant TP procesui analizuojama esamojo sprendinio p aplinka (mūsų atveju — $\Theta_2(p)$) ir atliekamas tas nedraudžiamas perėjimas, kuris labiausiai pagerina (sumažina) tikslo funkcijos z reikšmę. Jeigu nėra pagerinančių perėjimų, tai pasirenkamas tas, kuris mažiausiai pablogina TF reikšmę. Tam, kad būtų eliminuoti paieškos ciklai, atvirkštinis perėjimas, taip pat grįžimas į kitus neseniai aplankytus sprendinius turi būti uždraudžiamas apibrėžtam laikotarpiui. Draudimai saugomi tabu atmintyje, t. y. sąraše T, įsimenant paskutinius nagrinėtus sprendinius. Mūsų atveju tabu sąrašas T realizuotas kaip dvimatė $n \times n$ dydžio matrica. Šiuo atveju matricos elementas t_{ij} saugo esamos iteracijos numerį ir uždraudimo periodą (angl.

tabu tenure) h; tokiu būdu ši reikšmė nurodo, nuo kurios iteracijos vėl galima sukeisti perstatymo i-tąjį ir j-tąjį narius. Parametro h reikšmė mūsų ITP algoritme yra prilyginta $0{,}3n$. Elementų $p(i)$, $p(j)$ sukeitimas (perėjimas į sprendinį $p^{i,j}$) yra negalimas, jeigu reikšmė t_{ij} yra didesnė negu esamas paieškos iteracijos numeris. Draudimas yra anuliuojamas atsitiktiniais momentais su labai maža tikimybe α ($\alpha < 0{,}05$). (Tai leidžia truputį padidinti nedraudžiamų sprendinių kiekį ir ne taip apriboti galimas paieškos kryptis.) Draudimas taip pat yra ignoruojamas, kai patenkinama aspiracijos sąlyga (angl. *aspiration criterion*), t. y. po sukeitimo gaunamas sprendinys, kuris yra geresnis už iki šiol atliekant tabu paieškos procedūrą rastą geriausią sprendinį. Tabu sąrašas yra dinamiškai atnaujinamas vykdant algoritmą (kai tik esamas sprendinys pakeičiamas nauju).

Kartu su tabu sąrašu mūsų tabu paieškos procedūra dar naudoja papildomą atmintį Γ (antrinę atmintį-archyvą (angl. *secondary memory*)) (Dell'amico, Trubian, 1998). Šios atminties paskirtis yra išsaugoti aukštos kokybės sprendinius, kurie nors ir buvo įvertinti kaip labai geri, bet nebuvo pasirinkti. Konkrečiai sakant, į antrinę atmintį kiekvienos iteracijos metu įtraukiami sprendiniai, likę „antri" po aplinkos Θ_2 patikrinimo. Jeigu vykdant tabu paieškos procedūrą geriausias rastas sprendinys nesikeičia daugiau kaip $[\beta\tau]$ iteracijų, tai tabu sąrašas išvalomas ir paieška paleidžiama, pradedant nuo vieno iš „antrųjų" sprendinių iš antrinės atminties Γ (čia τ yra nustatytas TP procedūros vykdymo iteracijų skaičius, o β – parinktas tuščios eigos limito koeficientas, toks, jog $1 \le [\beta\tau] \le \tau$). Reikia pastebėti, jog archyve išsaugomos ir visos tikslo funkcijos pokyčių reikšmės (Δ), tai padidina atminties sąnaudas, bet skaičiavimų laikui tai labai didelės įtakos nedaro. TP algoritmo sudėtingumas išlieka proporcingas $O(n^2)$, o tokio archyvo panaudojimas, kaip rodo eksperimentų rezultatai, prisideda prie to, jog efektyvumas padidėja dėl to, jog sumažinamas neigiamas paieškos stagnacijos poveikis. Antrinė atmintis išvaloma, pasibaigus TP procedūrai.

TP procesas yra baigiamas, kai tik atliekamas iš anksto pasirinktas TP vykdymo iteracijų skaičius (τ). TP algoritmo detalizuotas formalus aprašymas pateiktas 1 pav.

---

procedure Tabu_Paieška; //tabu paieškos algoritmas KP uždaviniui
//duomenys: p – esamas sprendinys-perstatymas (kartu su atitinkamais tikslo funkcijos reikšmių pokyčiais (Δ)),
// n – uždavinio dydis (perstatymo dydis)
//rezultatai: $p^{\bullet}$ – geriausias rastas sprendinys (kartu su TF reikšmių pokyčiais)
//valdymo parametrai: τ – tabu paieškos vykdymo iteracijų skaičius, h – uždraudimo (tabu) periodas,
// α – tabu paieškos randomizavimo koeficientas, β – tuščios eigos limito koeficientas

---

begin
$p^{\bullet} := p$; $k := 1$; $k' := 1$; *archyvo_skaitiklis* $:= 0$; $L :=$; //L – tuščios eigos limitas
while ($k \le \tau$) **begin** //pagrindinis tabu paieškos iteracijų vykdymo ciklas
$delta'_{min} := \infty$; $delta''_{min} := \infty$; $u' := 1$; $v' := 2$;
//nagrinėjama sprendinio p aplinka $\Theta_2(p)$ ir ieškomas geriausias sprendinys, kuris nėra uždraustas (tabu)
for $i := 1$ **to** $n - 1$**do**

```
      for j := i + 1 to n>do begin
        delta := Δ(p^{i,j}, p);
        tabu := iif((t_ij ≥ k) and (random() ≥ α), TRUE, FALSE); aspiracija := iif(z(p) + del-
ta < z(p•), TRUE, FALSE);
        if ((delta < delta'_min) and (tabu = FALSE)) or (aspiracija = TRUE) then be-
gin
          if delta < delta'_min then begin
            delta''_min := delta'_min; u'' := u'; v'' := v'; delta'_min := delta; u' := i;
v' := j endif
          else if delta < delta''_min then begin delta''_min := delta; u'' := i; v'' := j
endif
        endif
      endfor;
    if delta''_min < ∞ then begin //„antrojo“ sprendinio įtraukimas į archyvą (antrinę
atmintį) Γ
      archyvo_skaitiklis := archyvo_skaitiklis + 1; Γ[archyvo_skaitiklis] ← p, Δ, u'', v'' en-
dif;
    if delta'_min < ∞ then begin
      p := p^{u',v'}; // esamas sprendinys pakeičiamas nauju sprendiniu
      perskaičiuoti tikslo funkcijos pokyčius , i, j = 1, ..., n;
      if z(p) < z(p•) then begin p• := p; k' := k endif; // išsaugomas geriausias
rastas sprendinys
      t_{u',v'}) := k + h // elementų p(u'), p(v') sukeitimas uždraudžiamas būsimoms h iteracijų
    endif;
    if (k – k' > L) and (k <τ – L) then begin //tabu paieškos paleidimas, panaudojant
sprendinį iš archyvo
      išvalyti tabu sąrašą T;
      atsitiktinio_išrinkimo_indeksas := random(archyvo_skaitiklis ∗ 0.8, archyvo_skaitiklis);
      p, Δ, u'', v'' ← Γ[atsitiktinio_išrinkimo_indeksas];
      p := p^{u'',v''}; perskaičiuoti tikslo funkcijos pokyčius , i, j = 1, ..., n;
      t^{u'',v''} := k + h; k' := k
    endif;
    k := k + 1
  endwhile;
  išvalyti tabu sąrašą T
  end.
```

Pastabos. 1. Tiesioginio vykdymo „if“ funkcija **iif**(x, y_1, y_2) grąžina y_1, jeigu x = TRUE, kitu atveju grąžinama y_2. 2. Funkcija **random**() generuoja (pseudo)atsitiktinį realųjį skaičių, patenkantį į intervalą [0, 1]. 3. Funkcija **random**(x_1, x_2) generuoja (pseudo)atsitiktinį sveikąjį skaičių iš intervalo [x_1, x_2].

1 pav. Tabu ***paieškos algoritmo aprašymas***

Iteracinis tabu paieškos algoritmas

Tabu paieškos algoritmo panaudojimas dar neužtikrina aukštos surastų sprendinių kokybės dėl jau minėto paieškos stagnacijos efekto pasireiškimo. Vienas iš būdų sumažinti paieškos

stagnavimo poveikį yra išplėsti standartinį TP algoritmą taip, kad jis būtų vykdomas ne vieną, bet keletą kartų. Taip gaunama iteratyvioji tabu paieška (ITP) (angl. *iterated tabu search*), kurios veikimo principo pagrindas yra daugkartinis TP algoritmo panaudojimas, iteratyviu būdu vykdant TP kartotinį procesą ir kombinuojant tabu paiešką su papildomomis sprendinių pertvarkymo, t. y. mutavimo, procedūromis[2]. Iteratyviajai tabu paieškai, savo ruožtu, taip pat gali būti taikomas daugkartinis panaudojimas; taip gaunamas 2 lygių iteratyviosios tabu paieškos algoritmas – savotiškas hierarchinis iteratyviosios tabu paieškos algoritmas. Tokio tipo algoritmo skiriamoji savybė yra ta, jog jis pasižymi savo struktūros hierarchiškumu, tiksliau, panašumumu į save (angl. *self-similarity*). Panašumas į save, kaip žinoma, yra turbūt vienas iš fundamentaliausių gamtos principų. Mūsų atveju dviejų lygių ITP algoritmą sudaro du panašūs į save struktūriniai lygiai. Aukštesniojo (2-ojo) lygmens algoritmas kreipiasi į žemesniojo (1-ojo) lygio algoritmą, kuris jau betarpiškai naudoja TP algoritmą. Šiuo atveju TP procedūra, galima įsivaizduoti, yra savotiškas iteracinio algoritmo „branduolys“. Būtent ši ir tik ši procedūra vykdo betarpišką sprendinio pagerinimą (atlieka paieškos intensifikavimą).

Du lygiai yra iš esmės tos pačios struktūros. Esamo lygio algoritmą sudaro trys esminiai komponentai: 1) žemesnio lygio algoritmo panaudojimas (iškvietimas); 2) sprendinio-kandidato parinkimas mutavimui; 3) atrinkto sprendinio mutavimas. Nors atskirų lygių struktūra yra išties paprasta, tačiau šių lygių tinkamas sujungimas leidžia žymiai padidinti paieškos efektyvumą. Aišku, paieškos laikas išauga, bet tai yra kompensuojama aukštesne rezultatų kokybe. Atkreiptinas dėmesys į tai, kad sprendinio mutavimas yra labai svarbus komponentas ITP algoritme ir jis yra reikalingas tam, kad būtų galima pereiti į kitas, nenagrinėtas sprendinių aibės (paieškos erdvės) sritis ir išvengti paieškos stagnacijos. Matyti, jog sprendinio mutavimas vykdomas kiekviename lygyje – skirtingai nuo „branduolio“ procedūros (TP procedūros), kuri atliekama tik žemiausiame lygyje. Taigi, iteracinio algoritmo atveju dominuoja paieškos diversifikavimo faktorius, o tai ir yra labai svarbu, vykdant paiešką didžiulėje sprendinių erdvėje su daug lokaliai optimalių sprendinių. Kas dėl sprendinio-kandidato parinkimo mutavimui, tai visada imamas paskutinis gautas (vis kitas) pagerintas sprendinys. Toks parinkimas leidžia išžvalgyti galimai didesnę sprendinių erdvės dalį.

Vieno ir dviejų lygių ITP algoritmų detalizuoti formalūs aprašymai pateikti 2 ir 3 pav.

---

procedure 1-Lygio_Iteracinė_Tabu_Paieška; //vieno lygio iteracinės tabu paieškos algoritmas KP uždaviniui
//duomenys: p – esamas sprendinys-perstatymas
//rezultatai: p^{∇} – geriausias rastas sprendinys-perstatymas (kartu su tikslo funkcijos reikšmių pokyčiais (Δ))
//valdymo parametras: Q_1 – 1 lygio tabu paieškos vykdymo iteracijų skaičius

---

begin
  $p^{\nabla} := p$;
  for $q_1 := 1$ **to** Q_1 **do begin**

---

[2] Panašus patobulinimas yra sukurtas ir klasikiniams lokaliosios paieškos algoritmams. Patobulinimas žinomas kaip iteratyvioji lokalioji paieška (angl. *iterated local search*) (Lourenco ir kt., 2002).

```
    vykdyti procedūrą Tabu_Paieška sprendiniui p, gauti (pagerintą)
sprendinį p•;
    if z(p•) < z(p∇) p∇ := p•; //įsimenamas geriausias rastas sprendinys (kartu su TF
pokyčiais)
    if q1 < Q1 then begin
       p := Sprendinio_Kandidato_Parinkimas_Mutavimui(p•, p∇);
       vykdyti mutavimo procedūrą sprendiniui p, grąžinti mutuotą
sprendinį p~;
       p := p~
    endif
  endfor
end.
```

2 pav. Vieno ***lygio iteracinės tabu paieškos algoritmo aprašymas***

```
procedure 2-jų_Lygių_Iteracinė_Tabu_Paieška; //dviejų lygių iteracinės tabu paieškos
algoritmas KP uždaviniui
//duomenys: p° – pradinis sprendinys-perstatymas
//rezultatai: p∇∇ – geriausias rastas sprendinys-perstatymas
//valdymo parametras: Q2 – 2 lygio tabu paieškos vykdymo iteracijų skaičius

begin
  generuoti atsitiktiniu būdu pradinį sprendinį-perstatymą p°;
  apskaičiuoti tikslo funkcijos pokyčius Δ((p°)i,j, p°), i, j = 1, ..., n;
  inicializuoti tabu sąrašą T;
  p := p°; p∇∇ := p°;
  for q2 := 1 to Q2 do begin
    vykdyti procedūrą 1-Lygio_Iteracinė_Tabu_Paieška sprendiniui p, gauti
(pagerintą) sprendinį p∇;
    if z(p∇) < z(p∇∇) p∇∇ := p∇; //įsimenamas geriausias rastas sprendinys
    if q2 < Q2 then begin
       p := Sprendinio_Kandidato_Parinkimas_Mutavimui(p∇, p∇∇);
       vykdyti mutavimo procedūrą sprendiniui p, grąžinti mutuotą
sprendinį p~;
       p := p~
    endif
  endfor
end.
```

3 pav. Dviejų ***lygių iteracinės tabu paieškos algoritmo aprašymas***

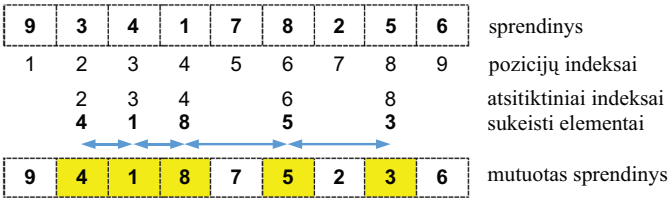
Mutavimo proceso bazinių variantų algoritminis realizavimas nėra labai sudėtingas. Teorinis pagrindas yra mutavimo operatorius φ, formaliai aprašytas sk. „Bazinės formuluotės“. Atsižvelgiant į mutavimo algoritminio realizavimo skirtumus (ypatumus), galima sudaryti įvairių tipų mutavimo procedūras.

Toliau trumpai aprašomos mūsų sudarytos ir tyrime naudotos mutavimo procedūros.

Atsitiktiniai poriniai elementų sukeitimai

Ši mutavimo procedūra yra labai nesudėtinga (žr. 4 pav.). Sukuriamas sprendinio-perstatymo p klonas $p\sim$. Perstatyme $p\sim$ atsitiktinai parenkamos pozicijos k_1, k_2, sugeneruojant du skirtingus (pseudo)atsitiktinius sveikuosius skaičius iš intervalo [1, n] pagal tolygiojo pasiskirstymo dėsnį. (Kad būtų išvengta pasikartojančių atsitiktinių skaičių generavimo, yra iš anksto sugeneruojamas perstatymas $s = (s(1), s(2), \ldots, s(n))$, sudarytas iš atsitiktinai sumaišytų sveikųjų skaičių nuo 1 iki n. Skaičių poros $(s(i), s(i + 1))$ tokiu būdu nurodo perstatymo $p\sim$ atsitiktinių pozicijų indeksų poras.) Pradedama nuo poros $(s(1), s(2))$, ir perstatymo $p\sim$ elementai, esantys šiose pozicijose, sukeičiami (iš $p\sim$ pereinama į $(p\sim)^{s(1),s(2)}$). Po to imama pora $(s(2), s(3))$ ir t. t. Tai kartojama $\mu - 1$ skaičių kartų; čia μ – algoritmo tyrėjo / vartotojo iš anksto nustatyta (norima) mutavimo laipsnio reikšmė. Mutavimo procedūros rezultatas – mutuotas sprendinys $p\sim$, tenkinantis sąlygą $\delta(p, p\sim) = \mu$. Aišku, kad formuojant mutuotą sprendinį-perstatymą yra užtikrinamas reikalavimas, jog gautame sprendinyje-perstatyme jokie elementai nesikartoja, t. y., $p\sim \in \Pi_n$. Mutavimo proceso stiprumas kontroliuojamas panaudojant parametrą μ (teoriškai $2 \le \mu \le n$).

Remiantis šiuo mutavimo algoritmo principu galima sudaryti ir daug įvairių kitų modifikuotų mutavimo procedūrų.



4 pav. ***Mutavimo procedūros pavyzdžio iliustracija (n = 9, μ = 5)***
(šiame pavyzdyje mutavimo procesas yra toks: elementas 3 keičiamas su elementu 4 (elementas 3 patenka į 3-iąją poziciją); 3 (esantis 3-iojoje pozicijoje) keičiamas su 1; 3 (esantis 4-ojoje pozicijoje) keičiamas su 8; 3 (esantis 6-ojoje pozicijoje) keičiamas su 5 (3 patenka į 8-ąją poziciją).)

Poriniai elementų sukeitimai panaudojant atsitiktinį raktą

Mutavimo procedūrą šiuo atveju sudaro du žingsniai: 1) atsitiktiniai poriniai elementų sukeitimai; 2) sukeistų elementų išmaišymas, panaudojant atsitiktinį raktą. Atsitiktinis raktas yra papildomas atsitiktinių indeksų nuo 1 iki μ masyvas (rinkinys $r = (r(1), r(2), \ldots, r(\mu))$). Atsitiktinio rakto reikšmės nurodo, kokie anksčiau sukeisti elementai su kokiais elementais maišomi. Taip bandoma gauti „giliau“ mutuotą sprendinį, daugiau randomizuotą elementų išmaišymą. Tuomet vykstant TP procesui ne taip paprasta grįžti į ankstesnę situaciją, tik atliekant porinius elementų sukeitimus.

*Mutavimas panaudojant priešines (opozicines) reik*šmes

Šioje mutavimo procedūroje sprendinio p klone $p\sim$ atsitiktinai parenkama pozicija, tarkime, k. Tuomet sprendinio $p\sim$ esama elemento reikšmė $e = p\sim (k)$ pakeičiama tokia priešine (opozicine) (angl. *opposition*) reikšme: $o = ((p\sim (k) + n/2 - 1) \bmod n) + 1$, čia mod žymi liekanos gavimo operaciją. Po šio sukeitimo taip pat turi būti pakeistas sprendinio elementas, kuris prieš tai buvo lygus o. Po abiejų sukeitimų $p\sim (k)$ tampa lygus o, $p\sim (l)$ – lygus e, l rodo sprendinio $p\sim$ elemento reikšmės, kuri buvo lygi o, indeksą.

Maksimalaus elementų atstumo mutavimas

Duotoje procedūroje sukeičiamų sprendinio elementų pozicijų indeksai generuojami taip, jog atstumas (λ) tarp tų pozicijų būtų galimai didesnis. Panaudota tokia sukeičiamų sprendinio elementų indeksų reikšmių k_l generavimo formulė: $k_l = [((\lambda q_l + \xi - 1) \bmod n) + 1]$, čia $\lambda = n/\mu$, ξ – (pseudo)atsitiktinis realusis skaičius iš intervalo [0, 1], $q_l = (q_{l-1} \bmod n) + 1$, $l = 1, 2, \ldots, \mu$; pradinė q_l reikšmė (q_0) – tai atsitiktinis sveikasis skaičius iš intervalo [1, n]. Tokiu būdu yra sugeneruojama μ atsitiktinių indeksų reikšmių (priminsime, kad μ yra mutavimo laipsnis). Šiuo atveju rekomenduojama, kad μ neviršytų $n/2$.

Modifikuotas atsitiktinių elementų sukeitimų mutavimas − I

Ši procedūra panaši į atsitiktinių porinių elementų sukeitimų mutavimo procedūrą. Generuojamas atsitiktinių realiųjų skaičių iš intervalo [0, 1] rinkinys (angl. *real-coded values*). Sugeneruoti skaičiai (kartu su juos atitinkančiais teigiamais indeksais (angl. *smallest positive values*)) išrikiuojami didėjimo tvarka. Išrikiuotus atsitiktinius skaičius atitinkantys išmaišyti indeksai ir nurodo, kurias sprendinio elementų poras reikia sukeisti (žr. sk. „Atsitiktiniai poriniai elementų sukeitimai“).

Modifikuotas atsitiktinių elementų sukeitimų mutavimas − II

Duota procedūra yra irgi panaši į procedūrą, aprašytą sk. „Atsitiktiniai poriniai elementų sukeitimai“. Procedūros tikslas – nuosekliai, vienas po kito generuojant atsitiktinius sveikuosius skaičius iš intervalo [0, 1], iš karto gauti sprendinio sukeičiamų elementų porų indeksus. Tačiau, kaip pastebėta, atsitiktiniai sveikieji skaičiai gali dubliuotis. Tam, kad nebūtų dubliavimosi, sugeneruoti atsitiktiniai sveikieji skaičiai surikiuojami (žr. sk. „Modifikuotas atsitiktinių elementų sukeitimų mutavimas – I“). Surikiuotus skaičius atitinkantys indeksai nurodo reikalingus sumaišyti elementus.

Kombinuoto mutavimo procedūra

Kombinuoto mutavimo procedūroje yra nuosekliai sujungtos dvi mutavimo procedūros. Iš pradžių nustatomi atsitiktiniai perstatymo elementų indeksai, kaip aprašyta sk. „Mo-

difikuotas atsitiktinių elementų sukeitimų mutavimas – II“. Po to nustatyti elementai yra mutuojami, panaudojant priešines reikšmes (žr. sk. „Mutavimas panaudojant priešines (opozicines) reikšmes“).

Randomizuotos lokalios paieškos mutavimas

Šiuo atveju iš pradžių vykdomi atsitiktiniai poriniai elementų sukeitimai (kaip aprašyta sk. „Atsitiktiniai poriniai elementų sukeitimai“). Po to atliekama randomizuota lokalioji paieška, t. y. nuosekliai nagrinėjamos atsitiktinai parinktos sprendinio elementų poros, bet elementų sukeitimai atliekami tik tuo atveju, jeigu po sukeitimo gaunamas geresnis sprendinys (tikslo funkcijos reikšmė sumažėja). Ši ir kitos toliau aprašomos mutavimo procedūros jau nėra tokios universalios (nepriklausančios nuo sprendžiamo uždavinio), kaip anksčiau aprašytos, nes turi būti įvertinama ir tikslo funkcija bei specifinės konkretaus sprendžiamo uždavinio savybės.

Randomizuotos dalinės lokalios paieškos mutavimas

Šioje procedūroje iš pradžių taip pat vykdomi atsitiktiniai elementų sukeitimai. Po to atliekama dalinė randomizuota lokalioji paieška. Šiuo atveju visiškai išnagrinėjama atsitiktiniu būdu sukonstruota, nedidelės apimties sprendinių aplinka. Vykdant paiešką šioje aplinkoje, vėlgi atliekami tik tie perėjimai tarp sprendinių, kurie pagerina tikslo funkcijos reikšmę.

Godžios adaptyvios paieškos mutavimas

Šios mutavimo procedūros principas yra kiek kitoks negu prieš tai nagrinėtų. Ypatumas yra tas, jog iš pradžių sprendinys savotiškai „dezintegruojamas“, po to rekonstruojamas. Sprendinio dezintegravimas reiškia, jog yra anuliuojama (pašalinama) dalis atsitiktinai parinktų sprendinio elementų, sprendinys tampa dalinis, po to bandoma dalinį sprendinį rekonstruoti į pilnąjį sprendinį ir taip, kad gautas sprendinys būtų su kuo mažesne tikslo funkcijos reikšme. Taigi, mutavimą sudaro dvi fazės: 1) sprendinio dezintegravimas, kuris yra atsitiktinis; 2) sprendinio rekonstravimas, kuris yra determinuotas (nes siekiama minimizuoti tikslo funkciją). Mūsų procedūroje pirmos fazės metu yra anuliuojama μ elementų, t. y. tiek, koks yra mutavimo laipsnis. Antroje fazėje taikomas godusis konstravimo algoritmas. Šis algoritmas iš visų galimų μ variantų atrenka tą, kuriam apskaičiuotas tikslo funkcijos įvertinimas (tam tikra tikslo funkcijos ribos reikšmė) yra minimalus. Mutavimo laipsnio μ dydis ($\mu \leq const$, $const \leq 7$), kad labai neišaugtų mutavimo procedūros atlikimo laikas.

Godžios randomizuotos adaptyvios paieškos mutavimas

Ši mutavimo procedūra primena prieš tai aprašytąją. Skirtumas yra tas, jog dalinio sprendinio rekonstravimo fazėje yra naudojamas godžios randomizuotos adaptyvios paieškos

algoritmas (angl. *greedy randomized adaptive search procedure – GRASP*) (Li ir kt., 1994), norint gauti pagerintą sprendinį.

Išsklaidytas (išsklaidytos tabu paieškos) mutavimas

Šiuo atveju atliekant mutavimo procedūrą yra tiesiog pertvarkytas tabu paieškos algoritmas, kuriame atliekami tik perėjimai tarp „antrųjų“ sprendinių. Atliekamas ribotas TP vykdymo iteracijų skaičius (μ iteracijų). Taigi, mutavimas tampa ne tiek atsitiktinis, kiek godus (kvazideterminuotas).

Pastebėtina, kad, atliekant visas mutavimo procedūras, pasikeičia sprendinio TF reikšmė, taigi yra labai svarbu efektyviai perskaičiuoti TF reikšmių pokyčius. Kaip žinoma, TF pokyčiai, sukeitus du elementus, apskaičiuojami panaudojant $O(n^2)$ operacijų. Kadangi visos aprašytos mutavimo procedūros atlieka apytikriai μ ar panašų (proporcingą) skaičių elementų sukeitimų, tai visų mūsų mutavimo procedūrų sudėtingumas yra proporcingas $O(\mu n^2)$.

Mutavimo laipsnis μ atlieka svarbų vaidmenį mutavimo procedūroje, o kartu ir ją naudojančiame iteraciniame tabu paieškos algoritme. Turi būti pasirenkamas tam tikras kompromisinis variantas tarp dviejų ekstremalių situacijų: 1) reikšmė μ yra (labai) maža; 2) reikšmė μ yra (labai) didelė (pvz., artima n). Pirmuoju atveju mutavimas negarantuotų pakankamai „nutolusio“ (nuo duotojo) sprendinio generavimo, o tai lemtų grįžimą atgal į ankstesnįjį lokalųjį optimumą po tabu paieškos etapo. Antruoju atveju paieška taptų labai panaši į daugkartinę paiešką, startuojant kiekvieną kartą tiesiog nuo atsitiktinių sprendinių ir prarandant paieškos metu lokaliai optimaliuose sprendiniuose sukauptą naudingą informaciją.

Kompiuteriniai eksperimentai

Buvo atlikti praktiniai kompiuteriniai-eksperimentiniai tyrimai, kurių tikslas – pademonstruoti aprašyto ITP algoritmo tinkamumą ir nustatyti galimai efektyviausias mutavimo procedūras, kaip vienus iš esminių ITP algoritmo sudedamųjų komponentų. Vykdant eksperimentinius tyrimus su ITP algoritmu ir mutavimo procedūromis, buvo pasinaudota kvadratinio paskirstymo uždavinio standartinių testavimo duomenų pavyzdžiais iš viešosios elektroninės KP uždavinio testavimo duomenų bibliotekos QAPLIB (Burkard ir kt., 1997). Eksperimentuose naudotas 3 GHz taktinio dažnio stacionarus asmeninis kompiuteris. Buvo naudota A. Misevičiaus ir D. Kuznecovaitės sudaryto iteracinio tabu paieškos algoritmo programinė realizacija C# programavimo kalba. ITP algoritmo ir visų mutavimo procedūrų programų tekstus galima rasti internete adresu: https://www.personalas.ktu.lt/~alfmise/.

Atliekant eksperimentus, ITP algoritmo bei kartu mutavimo procedūrų veikimo efektyvumo įvertinimo ir palyginimo kriterijus yra gaunamų sprendinių kokybės vidutinis santykinis procentinis nuokrypis nuo geriausios žinomos tikslo funkcijos reikšmės (GŽR) (angl. *best known value*). Vidutinis santykinis procentinis nuokrypis ($\overline{\theta}$) apskaičiuojamas

pagal formulę: $\overline{\theta} = 100(\overline{z} - z_{GŽR})/z_{GŽR}$ [%], čia $\overline{z}$ yra vidutinė tikslo funkcijos reikšmė, gauta atlikus 10 pakartotinių algoritmo vykdymų, $z_{GŽR}$ žymi geriausią žinomą tikslo funkcijos reikšmę, atitinkančią (pseudo)optimalų sprendinį iš bibliotekos QAPLIB.

Kompiuteriniuose eksperimentuose panaudoti šie bibliotekoje QAPLIB esantys testavimo duomenų pavyzdžiai: `tai10a, tai12a, tai15a, tai17a, tai20a, tai25a, tai30a, tai35a, tai40a, tai50a`. Šie testavimo duomenys yra sugeneruoti atsitiktiniu būdu ir laikytini sunkiausiai sprendžiamais pavyzdžiais (Taillard, 1991). Eksperimentuota su trimis skirtingomis mutavimo laipsnio reikšmėmis: $\mu = 5$, $\mu = 6$, $\mu = 7$.

Eksperimentuose buvo tirtos anksčiau aprašytos mutavimo procedūros:

1) atsitiktinių porinių elementų sukeitimų mutavimas – M1;
2) atsitiktinio rakto mutavimas – M2;
3) priešinių reikšmių mutavimas – M3;
4) maksimalaus atstumo mutavimas – M4;
5) modifikuotas atsitiktinių sukeitimų mutavimas (I) – M5;
6) modifikuotas atsitiktinių sukeitimų mutavimas (II) – M6;
7) kombinuotas mutavimas – M7;
8) randomizuotos lokalios paieškos mutavimas – M8;
9) randomizuotos dalinės lokalios paieškos mutavimas – M9;
10) godžios adaptyvios paieškos mutavimas – M10;
11) godžios randomizuotos adaptyvios paieškos mutavimas – M11;
12) išsklaidytas mutavimas – M12.

Atliktų eksperimentų rezultatai pateikti 1–5 lentelėse. Lentelėse parodytos gautų sprendinių vidutinių nuokrypių nuo (pseudo)optimalių sprendinių reikšmės ($\overline{\theta}$) visoms mutavimo procedūroms. Testavimo duomenų pavyzdžiams `tai10a`, `tai12a`, `tai15a` visos mutavimo procedūros įgalino pasiekti (pseudo)optimalius arba jiems labai artimus sprendinius. Testavimo duomenims `tai10a`, `tai12a`, `tai15a`, `tai17a`, `tai20a` vidutinis nuokrypis neviršija 1 proc., visiems duomenims – 1,9 proc. Visa tai liudija pakankamai aukštą ITP algoritmo ir mutavimo procedūrų efektyvumo laipsnį.

1 lentelė. ***Mutavimo procedūrų palyginimo rezultatai testavimo duomenų pavyzdžiams `tai10a`, `tai12a`***

	`tai10a`			`tai12a`		
	$\overline{\theta}$			$\overline{\theta}$		
	$\mu = 5$	$\mu = 6$	$\mu = 7$	$\mu = 5$	$\mu = 6$	$\mu = 7$
M1	0,000	0,000	0,000	0,000	0,000	0,000
M2	0,000	0,000	0,000	0,000	0,000	0,000
M3	0,000	0,000	0,000	0,000	0,000	0,000
M4	0,000	0,000	0,000	0,000	0,000	0,000
M5	0,000	0,000	0,000	0,000	0,000	0,000
M6	0,000	0,000	0,000	0,000	0,000	0,000

M7	0,000	0,000	0,000	0,000	0,000	0,000
M8	0,000	0,000	0,000	0,000	0,000	0,000
M9	0,000	0,000	0,000	0,000	0,000	0,000
M10	0,000	0,000	0,000	0,000	0,000	0,000
M11	0,000	0,000	0,000	0,000	0,000	0,000
M12	0,000	0,000	0,000	0,000	0,000	0,000

2 lentelė. ***Mutavimo procedūrų palyginimo rezultatai testavimo duomenų pavyzdžiams `tai15a`, `tai17a`***

	`tai15a`			`tai17a`		
	$\overline{\theta}$			$\overline{\theta}$		
	$\mu = 5$	$\mu = 6$	$\mu = 7$	$\mu = 5$	$\mu = 6$	$\mu = 7$
M1	0,000	0,000	0,000	0,708	0,273	0,075
M2	0,020	0,002	0,000	0,590	0,311	0,056
M3	0,159	0,135	0,080	0,355	0,324	0,129
M4	0,100	0,020	0,000	0,346	0,286	0,038
M5	0,040	0,000	0,020	0,456	0,165	0,188
M6	0,021	0,021	0,020	0,220	0,259	0,150
M7	0,100	0,021	0,200	0,465	0,534	0,339
M8	0,000	0,000	0,000	0,490	0,150	0,113
M9	0,000	0,000	0,000	0,268	0,113	0,093
M10	0,080	0,001	0,200	0,684	0,292	0,167
M11	0,080	0,020	0,042	0,695	0,688	0,449
M12	0,159	0,110	0,250	0,383	0,642	0,621

3 lentelė. ***Mutavimo procedūrų palyginimo rezultatai testavimo duomenų pavyzdžiams `tai20a`, `tai25a`***

	`tai20a`			`tai25a`		
	$\overline{\theta}$			$\overline{\theta}$		
	$\mu = 5$	$\mu = 6$	$\mu = 7$	$\mu = 5$	$\mu = 6$	$\mu = 7$
M1	0,522	0,624	0,566	1,045	0,.973	0,753
M2	0,505	0,272	0,648	0,857	0,883	0,899
M3	0,794	0,823	0,578	0,885	1,123	1,190
M4	0,605	0,685	0,441	1,006	0,849	0,722
M5	0,536	0,563	0,502	1,015	0,936	0,957
M6	0,689	0,627	0,509	1,097	1,092	0,688
M7	0,475	0,520	0,757	1,129	1,484	1,034
M8	0,569	0,373	0,345	1,067	0,843	0,718
M9	0,554	0,360	0,433	0,727	0,769	0,685

M10	0,659	0,441	0,506	1,122	1,151	1,049
M11	0,646	0,537	0,722	0,970	1,005	1,180
M12	0,567	0,835	0,773	1,216	1,329	1,160

4 lentelė. ***Mutavimo procedūrų palyginimo rezultatai testavimo duomenų pavyzdžiams `tai30a`, `tai35a`***

	tai30a			tai35a		
	$\overline{\theta}$			$\overline{\theta}$		
	$\mu = 5$	$\mu = 6$	$\mu = 7$	$\mu = 5$	$\mu = 6$	$\mu = 7$
M1	1,086	1,043	0,904	1,279	1,296	1,052
M2	1,362	0,891	0,892	1,278	1,195	1,158
M3	1,457	1,105	1,237	1,483	1,542	1,360
M4	1,239	1,118	0,984	1,465	1,159	1,031
M5	1,235	1,006	0,872	1,300	1,150	1,197
M6	1,200	0,964	0,938	1,072	1,204	1,024
M7	1,257	1,170	1,181	1,270	1,109	0,959
M8	1,091	0,927	0,980	1,234	1,152	1,194
M9	1,040	0,800	0,795	1,133	1,261	1,095
M10	1,202	1,002	1,089	1,151	1,260	1,275
M11	1,272	1,242	1,307	1,387	1,389	1,240
M12	0,763	1,119	1,220	1,243	1,216	1,221

5 lentelė. ***Mutavimo procedūrų palyginimo rezultatai testavimo duomenų pavyzdžiams `tai40a`, `tai50a`***

	tai40a			tai50a		
	$\overline{\theta}$			$\overline{\theta}$		
	$\mu = 5$	$\mu = 6$	$\mu = 7$	$\mu = 5$	$\mu = 6$	$\mu = 7$
M1	1,444	1,259	1,160	1,691	1,701	1,391
M2	1,315	1,468	1,417	1,525	1,550	1,687
M3	1,595	1,409	1,549	1,653	1,725	1,849
M4	1,485	1,323	1,227	1,574	1,732	1,704
M5	1,259	1,364	1,343	1,626	1,761	1,598
M6	1,312	1,475	1,170	1,614	1,765	1,427
M7	1,548	1,529	1,542	1,755	1,817	1,889
M8	1,247	1,355	1,256	1,828	1,634	1,526
M9	1,179	1,268	1,024	1,635	1,672	1,439
M10	1,419	1,406	1,393	1,735	1,563	1,697
M11	1,451	1,389	1,500	1,713	1,638	1,713
M12	1,298	1,273	1,477	1,387	1,673	1,770

Iš pateiktų palyginimo rezultatų matyti, jog randomizuotos dalinės lokalios paieškos mutavimo (M9) ir randomizuotos lokalios paieškos mutavimo (M8) panaudojimas leidžia pasiekti geresnius iteracinio tabu paieškos algoritmo rezultatus, palyginti su likusiomis kitomis mutavimo procedūromis. Pažymėtina, jog ir paprasta mutavimo procedūra (M1), besiremianti tik atsitiktiniais poriniais elementų sukeitimais, įgalina gauti geros kokybės sprendinius. Be to, M1 reikalauja, ko gero, mažiausiai centrinio procesoriaus laiko.

Matyti, kad mutavimo laipsnio reikšmės padidinimas nebūtinai pagerina gaunamų sprendinių kokybę. Gali būti, jog ši reikšmė yra susijusi su sprendžiamų pavyzdžių dydžiu . Šiai priklausomybei nustatyti reikalingi papildomi eksperimentiniai tyrimai.

Baigiamosios pastabos

Šiame straipsnyje aprašytas dviejų lygių iteracinis tabu paieškos (ITP) algoritmas kvadratinio paskirstymo uždavinio sprendimui. Pagrindinis dėmesys skirtas į ITP algoritmą integruotoms mutavimo procedūroms, kurių esminė paskirtis yra diversifikuoti sprendinius, gaunamus vykdant tabu paiešką, ir taip pagerinti paieškos efektyvumą, lyginant su tradiciniais TP algoritmais, nenaudojančiais mutavimo.

Atlikti kompiuteriniai eksperimentai su dvylika pasiūlytų įvairių tipų mutavimo procedūrų, pradedant universalizuotu, grynai atsitiktiniu mutavimu ir baigiant kombinuotomis (kvazideterminuotomis) mutavimo procedūromis, kurios yra gana specializuotos ir orientuotos į konkretų sprendžiamą KP uždavinį. Sudarytos mutavimo procedūros pasižymi lankstumu, mutavimo stiprumą galima valdyti pagal vartotojo (tyrėjo) poreikius. Atlikti eksperimentiniai tyrimai, panaudojant KP uždavinio testavimo pavyzdžius iš bibliotekos QAPLIB, rodo, kad dviejų lygių ITP algoritmas leidžia gauti aukštos kokybės sprendinius. Palyginus įvairių mutavimo procedūrų efektyvumo rezultatus, galima daryti išvadą, jog mutavimo procedūros, kuriose įvertinamos tikslo funkcijos reikšmės ir kuriose yra integruotos papildomos priemonės sprendiniui dalinai pagerinti, yra galimai pranašesnės už tas procedūros, kurios neatsižvelgia į konkretaus uždavinio specifiką.

Ateityje dviejų lygių iteracinis tabu paieškos algoritmas galėtų būti toliau tobulinamas, integruojant į jį ištirtas efektyviausias mutavimo procedūras.

Literatūra

ABDEL-BASSET, Mohamed; MANOGARAN, Gunsekaran; EL-SHAHAT, Doaa; MIRJALILI, Seyedali (2018). Integrating the Whale Algorithm with Tabu Search for Quadratic Assignment Problem: a New Approach for Locating Hospital Departments. *Applied Soft Computing*, vol. 73, p. 530–546. Prieiga per internetą: <https://doi.org/10.1016/j.asoc.2018.08.047>.

AHMED, Zakir H. (2015). A Multi-Parent Genetic Algorithm for the Quadratic Assignment Problem. *OPSEARCH*, vol. 52, p. 714–732. Prieiga per internetą: <https://doi.org/10.1007/s12597-015-0208-7>.

AKSAN, Yagmur; DOKEROGLU, Tansel; COSAR, Ahmet (2017). A Stagnation-aware Cooperative Parallel Breakout Local Search Algorithm for the Quadratic Assignment Problem. *Comput-*

ers & Industrial Engineering, vol. 103, p. 105–115. Prieiga per internetą: <https://doi.org/10.1016/j.cie.2016.11.023>.

BENLIC, Una; HAO, Jin-Kao (2013). Breakout Local Search for the Quadratic Assignment Problem. *Applied Mathematics and Computation*, vol. 219, p. 4800–4815. Prieiga per internetą: <https://doi.org/10.1016/j.amc.2012.10.106>.

BENLIC, Una; HAO, Jin-Kao (2015). Memetic Search for the Quadratic Assignment Problem. *Expert Systems with Applications*, vol. 42, p. 584–595. Prieiga per internetą: <https://doi.org/10.1016/j.eswa.2014.08.011>.

BURKARD, Rainer E.; KARISCH, Stefan E.; RENDL, Franz (1997). QAPLIB – a Quadratic Assignment Problem Library. *Journal of Global Optimization*, vol. 10, p. 391–403. Prieiga per internetą: <https://doi.org/10.1016/0377-2217(91)90197-4> [žr. taip pat prieigą per internetą: <http://anjos.mgi.polymtl.ca/qaplib/>].

ÇELA, Eranda (1998). *The Quadratic Assignment Problem: Theory and Algorithms*. Dordrecht: Kluwer. 304 p.

CHMIEL, Wojciech (2019). Evolutionary Algorithm Using Conditional Expectation Value for Quadratic Assignment Problem. *Swarm and Evolutionary Computation*, vol. 46, p. 1–27. Prieiga per internetą: <https://doi.org/10.1016/j.swevo.2019.01.004>.

CHMIEL, Wojciech; KWIECIEŃ, Joanna (2018). Quantum-inspired Evolutionary Approach for the Quadratic Assignment Problem. *Entropy*, vol. 20, p. 781. Prieiga per internetą: <https://doi.org/10.3390/e20100781>.

DELL'AMICO, Mauro; TRUBIAN, Marco (1998). Solution of Large Weighted Equicut Problems. *European Journal of Operational Research*, vol. 106, p. 500–521.

DOKEROGLU, Tansel; SEVINÇ, Ender; COSAR, Ahmet (2019). Artificial Bee Colony Optimization for the Quadratic Assignment Problem. *Applied Soft Computing*, vol. 76, p. 595–606. Prieiga per internetą: <https://doi.org/10.1016/j.asoc.2019.01.001>.

DREZNER, Zvi (2003). A New Genetic Algorithm for the Quadratic Assignment Problem. *INFORMS Journal on Computing*, vol. 15, p. 320–330.

EDELKAMP, Stefan; SCHRÖDL, Stefan (2012). *Heuristic Search. Theory and Applications*. Waltham: Morgan Kaufmann Publishers. 712 p.

GAMBARDELLA, Luca M.; TAILLARD, Éric D.; DORIGO, Marco (1999). Ant Colonies for the Quadratic Assignment Problem. *Journal of the Operational Research Society*, vol. 50, p. 167–176. https://doi.org/10.1057/palgrave.jors.2600676

GLOVER, Fred; LAGUNA, Manuel (1997). *Tabu Search*. Dordrecht: Kluwer. 382 p.

HAFIZ, Faizal M. F.; ABDENNOUR, Adel (2016). Particle Swarm Algorithm Variants for the Quadratic Assignment Problems - a Probabilistic Learning Approach. *Expert Systems with Applications*, vol. 44, p. 413–431. Prieiga per internetą: <https://doi.org/10.1016/j.eswa.2015.09.032>.

HANAN, Maurice; KURTZBERG, Jerome M. (1972). Placement Techniques. In Breuer, M. A. (ed.). *Design Automation of Digital Systems: Theory and Techniques*, vol. 1, Englwood Cliffs: Prentice-Hall, p. 213–282.

KOOPMANS, Tjalling C.; BECKMANN, Martin (1957). Assignment Problems and the Location of Economic Activities. *Econometrica*, vol. 25, p. 53–76. https://doi.org/10.2307/1907742

LI, Yong; PARDALOS, Panos M.; RESENDE, Mauricio G. C. (1994). A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem. In Pardalos, P. M.; Wolkowicz, H. (eds.). *Quadratic Assignment and Related Problems*. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 16, Providence: AMS, p. 237–261. https://doi.org/10.1090/dimacs/016/12

LOURENCO, Helena R.; MARTIN, Olivier C.; STÜTZLE, Thomas (2002). Iterated Local Search. In Glover, F.; Kochenberger, G. (eds.). *Handbook of Metaheuristics.* Norwell: Kluwer, p. 321–353.

MERZ, Peter; FREISLEBEN, Bernd (2000). Fitness Landscape Analysis and Memetic Algorithms for the Quadratic Assignment Problem. *IEEE Transactions on Evolutionary Computation*, vol. 4, p. 337–352. https://doi.org/10.1109/4235.887234

MICHALEWICZ, Zbigniew; FOGEL, David B. (2000). *How to Solve It: Modern Heuristics*. Berlin-Heidelberg: Springer. 467 p.

MISEVIČIUS, Alfonsas (2003). A Modified Simulated Annealing Algorithm for the Quadratic Assignment Problem. *Informatica*, vol. 14, p. 497–514.

MISEVICIUS, Alfonsas (2004). An Improved Hybrid Genetic Algorithm: New Results for the Quadratic Assignment Problem. *Knowledge-Based Systems*, vol. 17, p. 65–73. https://doi.org/10.1016/j.knosys.2004.03.001

MISEVICIUS, Alfonsas (2005). A Tabu Search Algorithm for the Quadratic Assignment Problem. *Computational Optimization and Applications*, vol. 30, p. 95–111. https://doi.org/10.1007/s10589-005-4562-x

SAHNI, Sartaj; GONZALEZ, Teofilo (1976). P-complete Approximation Problems. *Journal of ACM*, vol. 23, p. 555–565.

SHYLO, Volodymyr P. (2017). Solving the Quadratic Assignment Problem by the Repeated Iterated Tabu Search Method. *Cybernetics and Systems Analysis*, vol. 53, p. 308–311. Prieiga per internetą: <https://doi.org/10.1007/s10559-017-9930-x>.

SIARRY, Patrick (ed.). (2016). *Metaheuristics*. Cham: Springer. 489 p.

STÜTZLE, Thomas (2006). Iterated Local Search for the Quadratic Assignment Problem. *European Journal of Operational Research*, vol. 174, p. 1519–1539. Prieiga per internetą: <https://doi.org/10.1016/j.ejor.2005.01.066>.

TAILLARD, Éric D. (1991). Robust Taboo Search for the QAP. *Parallel Computing*, vol. 17, p. 443–455. https://doi.org/10.1016/s0167-8191(05)80147-4

TANG, Jing; LIM, Meng-Hiot; ONG, Yew S.; ER, Meng J. (2006). Parallel Memetic Algorithm with Selective Local Search for Large Scale Quadratic Assignment Problems. *International Journal of Innovative Computing, Information and Control*, vol. 2, p. 1399–1416. https://doi.org/10.1109/cec.2003.1299905

YANG, Xin-She (2010). *Nature-Inspired Metaheuristic Algorithms*. Frome: Luniver Press. 160 p.