KAUNO TECHNOLOGIJOS UNIVERSITETAS

DOVILĖ VERENĖ

GENETINIAI HIERARCHINIAI ALGORITMAI KVADRATINIO PASKIRSTYMO UŽDAVINIUI

Daktaro disertacija
Gamtos mokslai, informatika (N 009)

Kaunas, 2022

Disertacija rengta 2014–2021 ir 2021–2022 metais Kauno technologijos universiteto Informatikos fakultete Multimedijos inžinerijos katedroje.
Doktorantūros teisė Kauno technologijos universitetui suteikta kartu su Vytauto Didžiojo universitetu ir Vilniaus Gedimino technikos universitetu.

Disertacija ginama eksternu.

Mokslinis vadovas:
prof. dr. Alfonsas MISEVIČIUS (Kauno technologijos universitetas, gamtos mokslai, informatika, N 009).

Redagavo: anglų kalbos redaktorė Brigita Brasienė (Leidykla „Technologija“), lietuvių kalbos redaktorė Violeta Meiliūnaitė (leidykla „Technologija“)

Informatikos mokslo krypties disertacijos gynimo taryba:
prof. habil. dr. Rimantas BARAUSKAS (Kauno technologijos universitetas, gamtos mokslai, informatika, N 009) – **pirmininkas**;
prof. habil. dr. Gintautas DZEMYDA (Vilniaus universitetas, gamtos mokslai, informatika, N 009);
prof. dr. Vacius JUSAS (Kauno technologijos universitetas, gamtos mokslai, informatika, N 009);
prof. dr. Olga KURASOVA (Vilniaus universitetas, gamtos mokslai, informatikos inžinerija, T 007);
doc. dr. Agnė PAULAUSKAITĖ-TARASEVIČIENĖ (Kauno technologijos universitetas, gamtos mokslai, informatika, N 009).

Disertacija bus ginama viešame Informatikos mokslo krypties disertacijos gynimo tarybos posėdyje 2022 m. balandžio 19 d. 10 val. Kauno technologijos universiteto Disertacijų gynimo salėje.

Adresas: K. Donelaičio g. 73-403, 44249 Kaunas, Lietuva.
Tel. (370) 37 300 042; faks. (370) 37 324 144; el. paštas doktorantura@ktu.lt

Disertacija išsiųsta 2022 m. kovo 18 d.
Su disertacija galima susipažinti internetinėje svetainėje http://ktu.edu, Kauno technologijos universiteto bibliotekoje (K. Donelaičio g. 20, 44239 Kaunas, Lietuva), Vytauto Didžiojo universiteto bibliotekoje (K. Donelaičio g. 52, 44244 Kaunas, Lietuva) ir Vilniaus Gedimino technikos universiteto bibliotekoje (Saulėtekio al. 14, 10223 Vilnius, Lietuva).

KAUNAS UNIVERSITY OF TECHNOLOGY

DOVILĖ VERENĖ

GENETIC HIERARCHICAL ALGORITHMS FOR THE QUADRATIC ASSIGNMENT PROBLEM

Doctoral dissertation
Natural Sciences, Informatics (N 009)

Kaunas, 2022

This doctoral dissertation was prepared at Kaunas University of Technology, Faculty of Informatics, Department of Multimedia Engineering during the period of 2014–2021 and 2021–2022.
The doctoral right has been granted to Kaunas University of Technology together with Vytautas Magnus University and Vilnius Gediminas Technical University.

The dissertation is defended externally.

Scientific Supervisor:
Prof. Dr. Alfonsas MISEVIČIUS (Kaunas University of Technology, Natural Sciences, Informatics, N 009).

Edited by: English language editor Brigita Brasienė (Publishing House "Technologija"), Lithuanian language editor Violeta Meiliūnaitė (Publishing House "Technologija")

Dissertation Defence Board of Informatics Science Field:
Prof. Habil. Dr. Rimantas BARAUSKAS (Kaunas University of Technology, Natural Sciences, Informatics, N 009) – **chairman**;
Prof. Habil. Dr. Gintautas DZEMYDA (Vilnius University, Natural Sciences, Informatics, N 009);
Prof. Dr. Vacius JUSAS (Kaunas University of Technology, Natural Sciences, Informatics, N 009);
Prof. Dr. Olga KURASOVA (Vilnius University, Natural Sciences, Informatics Engineering, T 007);
Assoc. Prof. Dr. Agnė PAULAUSKAITĖ–TARASEVIČIENĖ (Kaunas University of Technology, Natural Sciences, Informatics, N 009).

The official defence of the dissertation will be held at 10 a.m. on 19 April, 2022 at the public meeting of Dissertation Defence Board of Informatics Science Field in Dissertation Defense Hall at Kaunas University of Technology.

Address: K. Donelaičio St. 73-403, Kaunas LT-44249, Lithuania.
Tel. no. (+370) 37 300 042; fax. (+370) 37 324 144; e-mail doktorantura@ktu.lt

Doctoral dissertation was sent on 18 March, 2022.
The doctoral dissertation is available on the internet at http://ktu.edu and at the library of Kaunas University of Technology (K. Donelaičio St. 20, 44239 Kaunas, Lithuania), Vytautas Magnus University library (K. Donelaičio St. 52, 44244 Kaunas, Lithuania) and Vilnius Gediminas Technical University Library (Saulėtekio al. 14, 10223 Vilnius, Lithuania).

TURINYS

PAVEIKSLĖLIŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

SANTRUMPŲ SĄRAŠAS

AF – aplinkos funkcija (angl. *neighbourhood function*)
CP – centrinis procesorius (angl. *CPU - central processing unit*)
EA – euristinis algoritmas (angl. *heuristic algorithm*)
GA – genetinis algoritmas (angl. *genetic algorithm*)
GH – genetinis-hierarchinis (angl. *genetic-hierarchical*)
GHA – genetinis hierarchinis algoritmas (angl. *genetic hierarchical algorithm*)
GO – globaliai optimalus (angl. *globally optimal*)
GŽR – geriausia žinoma (tikslo funkcijos) reikšmė (angl. *best known value* (*BKV*))
GŽS – geriausias žinomas sprendinys (angl. *best known solution* (*BKS*))
HGA – hibridinis genetinis algoritmas (angl. *hybrid genetic algorithm*)
HEA – hierarchinis euristinis algoritmas (angl. *hierarchical heuristic algorithm*)
HIP – hierarchinė iteratyvioji paieška (angl. *hierarchical iterated* (*iterative*) *search*)
HITP – hierarchinė iteratyvioji tabu paieška (angl. *hierarchical iterated tabu search*)
ILP – iteratyvioji lokalioji paieška (angl. *iterated local search*)
IP – iteratyvioji paieška (angl. *iterated* (*iterative*) *search*)
ITP – iteratyvioji tabu paieška (angl. *iterated tabu search*)
KO – kombinatorinis optimizavimas (angl. *combinatorial optimization*)
KP / KPU – kvadratinio paskirstymo / kvadratinio paskirstymo uždavinys (angl. *quadratic assignment / quadratic assignment problem* (*QAP*))
LO – lokalusis optimumas (lokaliai optimalus) (angl. *local optimum, locally optimal*)
LP – lokalioji paieška (angl. *local search*)
NP-sunkus uždavinys – uždavinys, priklausantis NP-sunkių uždavinių sudėtingumo klasei. Šią klasę sudaro uždaviniai, kurie išsprendžiami tiksliai su nedeterminuota polinomine (angl. *non-deterministic polynomial*) skaičiavimo mašina (Tiuringo mašina). Tokie uždaviniai yra eksponentinio sudėtingumo uždaviniai.
QAPLIB – kvadratinio paskirstymo uždavinio testinių pavyzdžių biblioteka (angl. *Quadratic Assignment Problem LIBrary*)
SA – sprendinių aibė (angl. *set of solutions*)
SD – sprendinio diversifikavimas (angl. *solution diversification*)
TF – tikslo funkcija (angl. *objective function*)
TP – tabu paieška (angl. *tabu search*)

1 ĮVADAS

Sudėtingų optimizavimo uždavinių sprendimas išlieka labai rimtas iššūkis. Tokiems uždaviniams priskirtinas kvadratinio paskirstymo (KP) uždavinys (angl. *quadratic assignment* (*problem*), kuris priklauso kombinatorinio (diskretinio) optimizavimo uždavinių klasei. KP uždaviniui nėra efektyvių tiksliųjų algoritmų, garantuojančių šio uždavinio išsprendimą per praktiškai priimtiną skaičiavimų laiką, todėl plačiai naudojami euristiniai algoritmai (angl. *heuristic algorithms*). Euristiniai algoritmai negarantuoja optimalaus sprendinio suradimo, tačiau jie leidžia surasti pakankamai geros kokybės sprendinius per vartotojui ar tyrėjui priimtiną laiką. Nežiūrint pasiektų daug žadančių rezultatų, euristiniai algoritmai tebėra daugelio mokslininkų intensyvių tyrinėjimų objektas. Euristiniai algoritmai yra nuolat tobulinami, vystomi, kuriami naujų tipų algoritmai, kurie leidžia efektyviau spręsti optimizavimo uždavinius ir įveikti iššūkius, kuriuos lemia žymiai išaugusi sprendžiamų uždavinių apimtis.

Vieni iš modernių euristinių algoritmų yra hierarchiniai euristiniai algoritmai (HEA) (angl. *hierarchical heuristic algorithms*). Tokių algoritmų veikimo principo esmė yra daugkartinis algoritmo panaudojimas, iteratyviu būdu vykdant paieškos procesą. Šių algoritmų skiriamoji savybė yra ta, jog jie pasižymi savo struktūros hierarchiškumu. Hierarchinį algoritmą sudaro keli struktūriniai lygmenys. Vieno (aukštesnio) lygmens algoritmas kreipiasi į kito (žemesnio) lygmens algoritmą, ir taip iki žemiausio. Atskiri lygmenys yra iš esmės tos pačios struktūros (yra savi-panašūs). Paprastai esamo lygmens algoritmą sudaro trys esminiai komponentai: 1) sprendinio parinkimas; 2) sprendinio lokalusis pagerinimas (optimizavimas); 3) sprendinio pertvarkymas. Ir nors, kaip matyti, HEA atskirų dalių, komponentų struktūra yra labai paprasta, tačiau šių dalių tinkamas apjungimas, kombinavimas leidžia išties žymiai padidinti šio tipo algoritmų veikimo efektyvumą. Būtent hierarchinio tipo euristiniai algoritmai ir yra nagrinėjami šiame disertaciniame darbe.

Tyrimų objektas

Disertacijos tyrimo objektas yra hierarchinio tipo euristiniai optimizavimo algoritmai, kuriems būdinga kelių lygmenų hierarchinė struktūra (architektūra) ir kurie yra orientuoti spręsti kvadratinio paskirstymo uždavinį.

Darbo tikslas

Darbo tikslas yra kompiuteriniais eksperimentais ištirti hierarchinės struktūros euristinių algoritmų, būtent genetinių algoritmų kvadratinio paskirstymo uždaviniui, efektyvumą, taip pat nustatyti, kurie šio tipo algoritmų variantai galėtų būti tinkamiausi sprendžiant KP uždavinį.

Darbo uždaviniai (tyrimo etapai)

Darbo tikslui pasiekti buvo iškelti tokie darbo uždaviniai.

1. Euristinių algoritmų, tarp jų genetinių algoritmų kvadratinio paskirstymo uždaviniui, žvalgomasis tyrimas, algoritmų veikimo efektyvumo teorinis-konceptualusis palyginimas.

2. Euristinių algoritmų, konkrečiai, genetinių algoritmų kvadratinio paskirstymo uždaviniui sudarymas, jų programinis realizavimas.

3. Kompiuterinių eksperimentų su įvairiomis sudarytomis genetinio algoritmo modifikacijomis ir patobulinimais atlikimas (algoritmų efektyvumo, vykdymo spartos ir pan. kompiuterinis ištyrimas).

4. Tinkamų (kokybės ir laiko atžvilgiu) algoritminių variantų (konfigūracijų) nustatymas, atsižvelgiant į gautų empirinių eksperimentinių tyrimų rezultatų kokybę.

Tyrimų metodika

Tyrimų metodika remiasi kombinatorinio optimizavimo teoriniais pagrindais, euristinių algoritmų projektavimo principais ir panaudojimu, taip pat algoritmų savybių ir efektyvumo analize.

Darbo mokslinis naujumas ir praktinė reikšmė

Darbo mokslinį naujumą nusako šie pagrindiniai aspektai.

1. Pasiūlytas naujos, originalios struktūros (architektūros) hierarchinis hibridinis genetinis algoritmas kvadratinio paskirstymo uždaviniui spręsti.

2. Sukurtas kelių (daugiau nei dviejų) hierarchinių lygmenų iteratyviosios tabu paieškos algoritmas KP uždaviniui, kuris gali būti naudojamas autonomiškai, taip pat integruotas į hierarchinio hibridinio genetinio algoritmo sudėtį kaip tinkama lokaliojo genetinio algoritmo sprendinių pagerinimo (optimizavimo) procedūra.

3. Genetinis algoritmas yra naujas tuo požiūriu, jog iki šiol būtent kvadratinio paskirstymo uždaviniui nebuvo naudotas genetinis algoritmas, kombinuojamas su hierarchine iteratyviąja tabu paieška. Tokio tipo algoritmas buvo naudotas kitam – pilkųjų šablonų formavimo – uždaviniui, kuris yra atskiras KP uždavinio atvejis.

4. Pasiūlyta originali universaliojo kryžminimo procedūra (algoritmas).

Darbo praktinė vertė yra ta, jog sudarytų algoritmų pagrindu gali būti realizuota kompiuterinė programa, kuri būtų išbandoma, sprendžiant realius, praktinius KP uždavinio pavyzdžius.

Ginamieji teiginiai

1. Sukurti hierarchiniai genetinis ir iteratyviosios tabu paieškos algoritmai KP uždaviniui leidžia per priimtiną skaičiavimų laiką gauti aukštos kokybės KP uždavinio sprendinius.

2. Nustatyta, jog sudarytas kombinuotas hierarchinis genetinis-iteratyviosios tabu paieškos algoritmas yra pranašesnis už kitus euristinius algoritmus didelei daliai tirtų KP uždavinio pavyzdžių vykdymo laiko ir sprendinių kokybės požiūriu. Parodyta, kad daugeliu atvejų hierarchinis-genetinis algoritmas pranoksta konkrečiai hibridinį genetinį-tabu paieškos algoritmą, memetinį algoritmą, dažnų pėdsakų algoritmą, o šie algoritmai priklauso efektyviausių euristinių algoritmų KP uždaviniui grupei.

Darbo rezultatų aprobavimas

Disertacijoje pateikti rezultatai publikuoti dviejuose mokslo darbuose: abu straipsniai paskelbti Web of Science sąrašo žurnaluose, turinčiuose cituojamumo indeksą. Rezultatai taip pat pristatyti dviejose tarptautinėse mokslinėse konferencijose.

Disertacijos struktūra

Disertaciją sudaro įvadas, trys skyriai, baigiamosios išvados, literatūros šaltinių sąrašas, autorės publikacijų sąrašas disertacijos tema. Pirmame skyriuje pateikiama literatūros apžvalga. Antrame – suformuluotas kvadratinio paskirstymo uždavinys ir pristatomi sudaryti algoritmai kvadratinio paskirstymo uždaviniui. Trečiame skyriuje aprašomi kompiuteriniai eksperimentai su įvairiais algoritmų variantais, pateikiami gauti eksperimentų rezultatai, sprendžiant KP uždavinio testinius pavyzdžius.

2 LITERATŪROS APŽVALGA

2.1 Optimizavimo uždaviniai

Iš pradžių formaliai apibrėžiame kvadratinio paskirstymo (KP) uždavinį (angl. *quadratic assignment problem*). Šis uždavinys yra vienas iš sudėtingų kombinatorinio optimizavimo uždavinių. KP uždavinys aprašomas taip (taip pat žr. 3 sk.). Duotos matricos $\boldsymbol{A} = (a_{ij})_{n\times n}$ ir $\boldsymbol{B} = (b_{kl})_{n\times n}$ bei aibė Π_n, kurią sudaro natūrinių skaičių nuo 1 iki n perstatymai. (Čia kiekvieną perstatymą sudaro nesikartojantys skaičiai $p(1),\ p(2),\ \ldots,\ p(n), p(i) \in \{1,\ 2,\ \ldots,\ n\}$.) Reikia surasti perstatymą $p_{opt} \in \Pi_n$ ir tokį, kad $p_{opt} = \arg\min_{p\in\Pi_n} z(p)$; čia z yra KPU tikslo funkcija ($z(p) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}b_{p(i)p(j)}$), o p atlieka KP uždavinio sprendinių vaidmenį.

Traktuojant labai apibendrintai, optimizavimo uždavinys gali būti apibrėžiamas kaip tikslas surasti geriausią nepriklausomų dydžių – kintamųjų – reikšmių konfigūraciją. Optimizavimo uždaviniai, kuriuose operuojama 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 (Du ir kt., 2013).

Formaliai kombinatorinio optimizavimo uždavinį galima aprašyti per porą $(S,\ f)$ (Blum, Roli, 2003). *S* yra (leistinųjų) sprendinių aibė (SA) (angl. *set of feasible solutions*). Nemažą KO uždavinių dalį sudaro uždaviniai, kuriuose kintamųjų reikšmių tipas yra perstatymas. Perstatymas, kaip žinoma, yra 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 sudaro visi įmanomi natūrinių skaičių nuo 1 iki n perstatymai, t. y.:

$$S = \begin{Bmatrix} s|s = \big(s(1), s(2), \ldots, s(n)\big), s(i) \in \{1,2,\ldots,n\}\ \forall i\ 1 \le i \le n, \\ s(i) \neq s(j)\ \forall i,j\ 1 \le i,j \le n, i \neq j \end{Bmatrix}; \qquad (1)$$

čia s žymi perstatymą (sprendinį-perstatymą), o $s(i)$ – perstatymo s i–tąjį elementą (narį); n yra uždavinio apimtis.

Optimizavimo uždaviniui pilnai apibrėžti turi būti nurodomas tam tikras sprendinių vertės matas, kiekvienam sprendiniui įgaunantis kurią nors skaitinę reikšmę. Kitaip tariant, turi būti apibrėžta funkcija f (tai ir yra antrasis poros $(S,\ f)$ narys), kurios apibrėžimo sritis yra aibė S, o reikšmių aibė – realiųjų (sveikųjų) skaičių aibė. Ši funkcija vadinama optimizavimo uždavinio tikslo funkcija (TF) (angl. *objective function*).

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

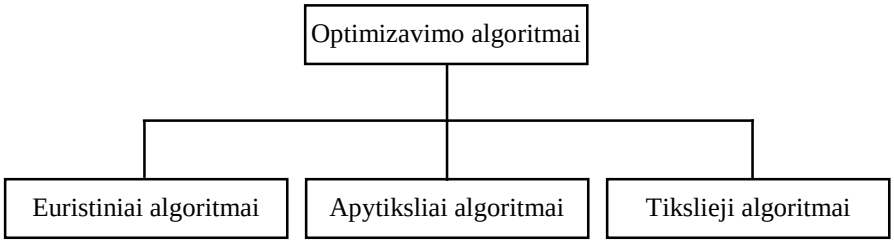
$$s_{opt} \in S_{opt} = \{s^{\nabla}|s^{\nabla} = \operatorname{argmin}_{s\in S} f(s)\}. \qquad (2)$$

Sprendinys s_{opt}, kuris minimizuoja tikslo funkciją f, vadinamas uždavinio (S, f) (globaliai) optimaliu (GO) sprendiniu.

Sprendžiant KO uždavinius, labai svarbi yra aplinkos funkcija (AF). Aplinkos funkciją Θ galima apibrėžti taip: $\Theta : S \to 2^S$ (čia 2^S žymi aibės S visų poaibių aibę). Naudojant šią funkciją sprendiniui $s \in S$ priskiriama aibė $\Theta(s) \subseteq S$ – sprendinio s aplinka (angl. *neighbourhood*), dar vadinama aplinkinių sprendinių (arba „sprendinių-kaimynų“) aibe. Konkrečios aplinkos pavyzdys yra 2-aplinka (žymima Θ_2), kuri dar žinoma kaip porinių sukeitimų aplinka. Turint AF, galima apibrėžti lokaliai optimalaus sprendinio (lokaliojo optimumo) sąvoką. Pažymėkime $\Delta_{s,s'} = f(s') - f(s)$ tikslo funkcijos reikšmių pokytį, gautą perėjus iš sprendinio s į sprendinį s'. Tuomet sprendinys s yra lokaliai optimalus aplinkos Θ atžvilgiu, jeigu kiekvienam sprendiniui s' iš aplinkos $\Theta(s)$ galioja $\Delta_{s,s'} \geq 0$ (visi TF pokyčiai yra teigiamai).

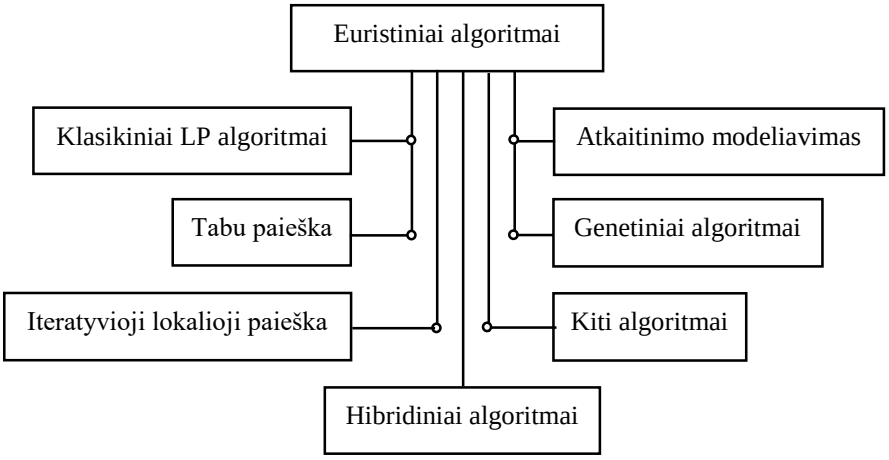
2.2 Optimizavimo uždavinių sprendimo euristiniai algoritmai

Algoritmų KO uždaviniams spręsti visuma yra skirstoma į dideles savarankiškas kategorijas (žr. toliau pateiktą schemą (1 pav.)).



1 pav. Optimizavimo algoritmų klasifikavimo schema

Euristiniai algoritmai (EA) (angl. *heuristic algorithms*) yra tokie optimizavimo uždavinių sprendimo metodai, kuriais siekiama surasti aukštos kokybės, bet nebūtinai optimalius sprendinius per priimtiną laiką (Siarry, 2016). Euristiniai algoritmai sprendžiamiems uždaviniams paprastai suranda tik nuo optimumo nutolusius sprendinius – lokaliai optimalius sprendinius. Todėl šie algoritmai dažnai vadinami lokaliosios paieškos (LP) algoritmais arba tiesiog lokaliąja paieška (angl. *local search*) (Aarts, Lenstra, 1997).



2 pav. Pagrindinių euristinių algoritmų klasifikavimo schema

Euristiniai algoritmai neužtikrina sprendinio optimalumo; nėra netgi garantuojama, kad surastas sprendinys bus „nutolęs“ nuo optimumo ne daugiau kaip tam tikras fiksuotas atstumas. Tuo euristiniai algoritmai skiriasi nuo apytikslių (aproksimacinių) algoritmų (angl. *approximate algorithms*), 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 (angl. *exact algorithms*). Pastarieji garantuoja optimalaus sprendinio suradimą; tiesa, laikas, reikalingas optimumui pasiekti, dažnai yra susietas su uždavinio apimtimi eksponentine priklausomybe (Garey, Johnson, 1979). Tuo tarpu euristiniai algoritmai pasižymi polinominiu paieškos laiko priklausomybės nuo uždavinio apimties pobūdžiu, tačiau šiuo atveju prarandama optimumo suradimo garantija – sprendinių tikslumas aukojamas vardan paieškos laiko.

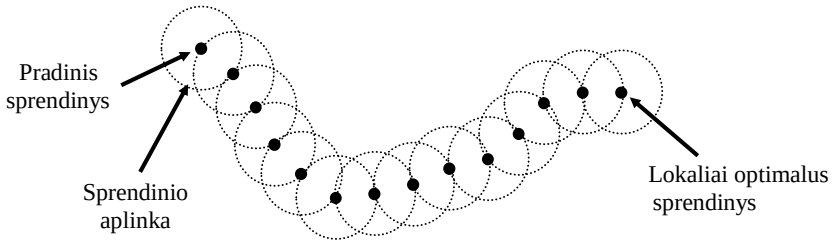
Nagrinėjant euristinius algoritmus susiduriama su jų didele įvairove. Yra keliasdešimt euristinių algoritmų grupių, o bendras EA spektras apima šimtus algoritmų variantų, pradedant elementariomis klasikinėmis godžiosios paieškos procedūromis ir baigiant labai jau intelektualizuotais, save apmokančiais metodais (Osman, Kelly, 1996; Salhi, 1998; Michalewicz, Fogel, 2000; Ribeiro, Hansen, 2001; Hoos, Stützle, 2004; Talbi, 2009; 2013; Yang, 2010(a); 2010(b); Zäpfel ir kt., 2010; Brownlee, 2011; Edelkamp, Schrödl, 2012; Boussaïd ir kt. 2013; Simon, 2013; Siarry, 2016; Sörensen ir kt., 2018; Hussain ir kt., 2019). Kai kurios pagrindinės euristinių algoritmų grupės pateiktos 2 pav.

2.3 Kvadratinio paskirstymo uždavinio sprendimo euristiniai algoritmai

Šiame skyriuje apžvelgiami euristiniai algoritmai, kurie naudojami būtent kvadratinio paskirstymo uždavinio sprendimui (žr. taip pat (Loiola ir kt., 2007; Drezner, 2015; Achary ir kt., 2021)).

2.3.1 Lokalioji paieška (angl. *local search*)

Klasikiniai euristiniai algoritmai yra lokaliai optimalių sprendinių paieškos algoritmai ir yra grindžiami „nusileidimo“ (godžiosios paieškos) metodologija (Vaessens ir kt., 1998; Voß ir kt., 1998). Nusileidimo algoritmai yra santykinai labai paprasti. Išskiriamos dvi svarbios nusileidimo strategijos: a) greičiausias nusileidimas; b) staigiausias nusileidimas. Abiem atvejais paieška pradedama nuo kurio nors pradinio sprendinio. Toliau paieškos procesas vykdomas nuosekliai atliekant sprendinių transformacijas ir pereinant nuo vieno sprendinio s prie kito sprendinio s' iš esamo sprendinio aplinkos $\Theta(s)$ (žr. 3 pav.). Abiejų tipų algoritmuose sprendimas, ar pakeisti esamą sprendinį nauju ar ne, yra determinuotas. Jis teigiamas tik tada, kai naujasis sprendinys yra geresnis už esamąjį (kas reiškia, jog TF reikšmių, apskaičiuotų naujam ir esamam sprendiniui, pokytis $\Delta f = f(s') - f(s)$ yra neigiamas). Skirtumas tarp strategijų yra tas, kad pirmuoju atveju pereinama prie pirmojo sutikto geresnio sprendinio iš esamos aplinkos, o antruoju – prie geriausio sprendinio. Paieška tęsiama tol, kol esamas sprendinys s tampa lokaliai optimalus.



3 pav. Paieškos „trajektorijos“ iliustracija
[panaudota iliustracija iš (Milinis, 2005)]

Lokaliosios paieškos euristiniai algoritmai savo efektyvumu nusileidžia kitiems euristiniams algoritmams, visų pirma, dėl paieškos ribotumo; t. y. apsiribojama tik vieno lokaliai optimalaus sprendinio suradimu. Toks surastas sprendinys, nors ir yra lokaliai optimizuotas, gali visai nebūti geros kokybės. Dar daugiau, galima netgi prognozuoti, kad toks sprendinys tikėtina kaip tik bus nepakankamos kokybės – juk tai tik vienas galimas lokaliai optimalus sprendinys iš didžiulio LO skaičiaus.

Kalbant apie kvadratinio paskirstymo uždavinį, LP algoritmai intensyviai naudoti šiam uždaviniui ankstyvuoju euristinių algoritmų taikymo laikotarpiu (apie 1960–1980 m.) (Armour, Buffa, 1963; Buffa ir kt., 1964; Murtagh ir kt., 1982). Tokie algoritmai dar vadinami klasikiniais lokaliosios paieškos algoritmais. Efektyvumui padidinti naudoti patobulinti LP algoritmai (Murthy ir kt., 1992; Pardalos ir kt., 1993; Angel, Zissimopoulos, 1998), taip pat ir lokalioji paieška su pertraukimais (angl. *breakout local search*) (Benlic, Hao, 2013; Aksan ir kt., 2017).

2.3.2 Atkaitinimo modeliavimas (angl. *simulated annealing*)

Atkaitinimo modeliavimo (AM) algoritmo ištakos slypi statistinėje mechanikoje, o tiksliau – energetinių procesų, vykstančių sistemose, sudarytose iš didelio skaičiaus dalelių, imitavime (Aarts, Korst, 1989; Aarts, Korst, 2001). Šio

imitavimo esmė ta, jog iš pradžių sistema pervedama į didelės energijos būseną, o po to, palaipsniui mažinant energiją, stengiamasi pasiekti būseną, atitinkančią žemiausią energetinį lygmenį – tarsi kūnas būtų įkaitintas iki pakankamai aukštos temperatūros, o paskui, mažinant temperatūrą, jis būtų savotiškai užgrūdinamas. AM pradininkai buvo Metropolis, Rozenbluth'as ir kt. (1953), kurie pasinaudojo eksponentinio pasiskirstymo funkcija apibrėždami tikimybę, kad sistema, įvykus joje tam tikram pasikeitimui, pereis iš vieno energijos lygio (E_1) į kitą (E_2), kai temperatūra lygi t. Ta tikimybė lygi 1, kai $\Delta E < 0$, ir $e^{-\Delta E/Ct}$, kai $\Delta E \geq 0$; čia $\Delta E = E_2 - E_1$, o C – konstanta. Cerný ir Kirkpatrick'as su bendraautoriais buvo pirmieji, pritaikę AM metodą sprendžiant KO uždavinius (Cerný, 1982; Kirkpatrick ir kt., 1983).

AM algoritmų veikimo principas yra gana paprastas. Pradedama nuo atsitiktiniu (ar kitu) būdu sukonstruoto sprendinio; iš esamo sprendinio s aplinkos parenkamas sprendinys s' ir apskaičiuojamas TF skirtumas $\Delta f = f(s') - f(s)$. Sprendimo taisyklė yra tokia: jeigu TF 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 $P(\Delta f) = e^{-\Delta f/t}$ (čia t yra einamoji temperatūra). Procesas tęsiamas tol, kol patenkinama algoritmo baigimo sąlyga.

Kalbant apie KP uždavinį, atkaitinimo modeliavimo algoritmai leido pasiekti geresnės kokybės rezultatus, lyginant su lokaliosios paieškos algoritmais. Tai pasakytina tiek apie ankstyvuosius AM algoritmų variantus (Burkard, Rendl, 1984; Wilhelm, Ward, 1987; Connolly, 1990), tiek patobulintas AM algoritmų modifikacijas (Bölte, Thonemann, 1996; Misevičius, 2003; Paul, 2011).

Bene pagrindinis atkaitinimo modeliavimo algoritmų silpnumas yra stochastinis (chaotinis) algoritmų pobūdis, taip pat jautrumas pradinėms sąlygoms. Šiuo atveju algoritmų vykdymą labai įtakoja inicializavimo pobūdis, taip pat valdymo parametrų (pradinės) reikšmės, kurias nėra taip paprasta tinkamai sukalibruoti.

2.3.3 Tabu paieška (angl. *tabu search*)

Tabu paieškos (TP) metodas buvo pasiūlytas F. Gloverio, P. Hanseno ir B. Jaumardo (Glover, 1989; 1990; Hansen, Jaumard, 1987). Šis metodas remiasi išplėsta lokaliąja paieška. Skirtingai negu įprasti klasikiniai algoritmai, kurie apsiriboja lokaliai optimalaus sprendinio suradimu nagrinėjamo sprendinio aplinkoje, TP pagrįsti algoritmai tęsia paiešką ir tuo atveju, kai surandamas lokaliai optimalus sprendinys.

Pagrindinė TP idėja yra leidimas atlikti perėjimus net ir tais atvejais, kai naujasis sprendinys iš duotojo sprendinio aplinkos nėra geresnis už duotąjį sprendinį. Tačiau kai kurie perėjimai turi būti draudžiami, t. y. tabu paieška yra pagrįsta draudimų metodologija: draudimai būtini tam, kad neleistų grįžti į jau nagrinėtus sprendinius. Paieškos procesas pradedamas nuo pradinio sprendinio s iš aibės S, po to perėjimai iš vieno sprendinio į kitą vykdomi iteraciniu būdu. Paieškos eigoje analizuojama einamojo sprendinio s aplinkos sprendinių aibė $\Theta(s)$ ir pereinama į tą sprendinį $s' \in \Theta(s)$, kuriam tikslo funkcijos f reikšmė yra mažiausia. Tačiau perėjimas gali būti atliekamas ir tuo atveju, kai TF pokytis yra teigiamas, t. y. perėjimas pablogina einamąją tikslo funkcijos reikšmę. Taigi, pasirenkamas tas perėjimas, kuris mažiausiai

pablogina TF reikšmę. Grįžimas į anksčiau aplankytą sprendinį uždraudžiamas tam tikram laikotarpiui – kad būtų išvengta ciklinimosi. Tokiu būdu nagrinėtieji sprendiniai tam tikrais momentais tampa „tabu“: jie įtraukiami į specialų sąrašą – tabu sąrašą T, kurio dydis $h = |T|$. Taigi, perėjimas į sprendinį $s' \in \Theta(s)$ yra draudžiamas, jeigu tas sprendinys arba atitinkamas perėjimas duotu metu yra sąraše T.

Tiesmukiškas perėjimų draudimas gali sumažinti paieškos našumą. Dėl šios priežasties naudojamas ir aspiracijos kriterijus (angl. *aspiration criterion*). Jo pagalba galima anuliuoti esamą tabu būseną, esant tam tikroms palankioms aplinkybėms. Vienas iš aspiracijos kriterijų yra toks: perėjimas iš sprendinio s į sprendinį s' leidžiamas, jeigu tenkinama sąlyga $f(s') < f(s^*)$, kur s^* yra geriausias iki duotojo momento surastas sprendinys. Tokiu būdu tipinė TP algoritmuose naudojama sprendimo priėmimo taisyklė yra tokia: s pakeičiamas s' su sąlyga, kad: $f(s') < f(s^*)$ arba ($s' = \arg\min_{s'' \in \Theta(s)} f(s'')$ ir s' nėra tabu) (Glover, Laguna, 1997).

Tabu paieškos algoritmų negatyvus aspektas yra tas, jog algoritmų vykdymui yra reikalingi papildomi atminties resursai – kas yra priešingybė, pvz., atkaitinimo modeliavimo algoritmams, kurie, nenaudoja kokios nors (papildomos) atminties. Kitas trūkumas yra galimas paieškos proceso ciklinimasis ir stagnacija. Taip pat pastebėtina, kad TP algoritmai naudoja įvairius valdymo parametrus, kuriuos reikia tinkamai sureguliuoti.

Nežiūrint to, tabu paieškos principo panaudojimas įgalino sukurti produktyvesnius algoritmus KP uždaviniui. Greito veikimo TP algoritmas, sukurtas 1991 m. (Taillard, 1991), dar ir dabar laikytinas vienu sėkmingiausių nepopuliacinių euristinių algoritmų KP uždaviniui. Nuo to laiko sukurta nemažai pagerintų TP algoritmų variantų: reaktyvioji tabu paieška (Battiti, Tecchiolli, 1994), išplėstinė tabu paieška (Misevicius, 2005), tabu paieška su aparatūriniu pagreitinimu (Zhu ir kt., 2010), kartotinio vykdymo tabu paieška (Shylo, 2017; Sergienko ir kt., 2020), lygiagreti tabu paieška (Abdelkafi ir kt., 2019), kiti variantai (Czapiński, 2013).

2.3.4 Genetiniai algoritmai (angl. *genetic algorithms*)

Genetinių algoritmų (GA) ir jų sudarymo principų pradininku buvo Holland‘as (Holland, 1975). Genetinių algoritmų veikimas yra pagrįstas evoliucijos, vykstančios gyvojoje gamtoje, imitavimu, ir pavadindami šiuos algoritmus natūraliosios atrankos virtualiąja kopija nedaug tesuklystume. Pagrindinės sąvokos, kurios naudojamos modeliuojant evoliucijos procesus, yra individas ir populiacija. Individas yra tam tikras elementarus vienetas. Individų grupė sudaro populiaciją. Svarbus dalykas yra ir „individo tinkamumas“ – savotiška individo vertė. Vertingesnis yra tas individas, kuris sugeba geriausiai prisitaikyti, pvz., palikti didesnį palikuonių skaičių. Optimizavime vietoje sąvokų „individas“, „populiacija“, „individo tinkamumas“ naudojamos įprastos sąvokos: individą atitinka atskiras sprendinys, populiaciją – sprendinių rinkinys, o individo vertė yra asocijuojama su tikslo funkcijos reikšme.

Pagrindinės GA savybės yra tokios. Pirma, operuojama su sprendinių populiacija – tai vienas iš esminių GA skiriamųjų požymių. Antra, atrenkant apdorojimui atskirus sprendinius iš sprendinių populiacijos, pirmenybė teikiama

tiems sprendiniams, kurie turi geresnes TF reikšmes. Trečia, naudojamas tam tikras mechanizmas, skirtas formuoti naujiems sprendiniams, kombinuojant turimus sprendinius. Ketvirta, esant reikalui, panaudojamas sprendinių randomizuoto pertvarkymo mechanizmas. Penkta, populiacija nuolat atnaujinama, pašalinant iš jos, pvz., blogesnius sprendinius. Formalizuojant GA aprašymą, anksčiau nurodytos savybės susiejamos su atitinkamomis procedūromis (Goldberg, 1989; Reeves, Rowe, 2001; Haupt, Haupt, 2004; Sivanandam, Deepa, 2008). Taip antroji savybė susiejama su atrinkimo procedūra, trečioji savybė vadinama kryžminimu, ketvirtoji – mutavimu, penktoji – populiacijos atnaujinimu.

Genetinių algoritmų (galimas) trūkumas yra pernelyg didelis algoritmų konvergavimo laikas, taip pat priešlaikinis konvergavimas bei populiacijos įvairovės praradimas ir to sąlygota paieškos proceso stagnacija.

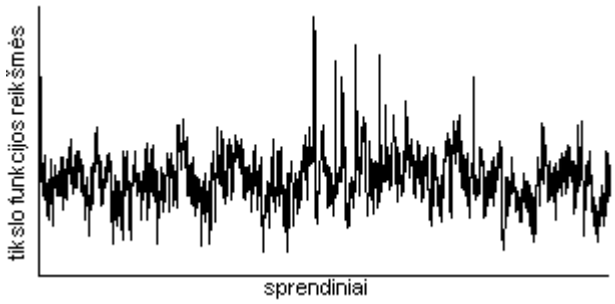
Genetinių algoritmų efektyvumas žymiai padidėja, kai šie algoritmai hibridizuojami su kitais algoritmais (Moscato, 1999).

GA savybės yra labai tinkamos sprendžiant KP uždavinį (Tate, Smith, 1995). Tyrėjų yra nustatyta, kad genetiniai algoritmai, o būtent hibridiniai genetiniai algoritmai yra perspektyviausi euristiniai algoritmai KP uždaviniui (tarp jų, paminėtini: godusis genetinis algoritmas (Ahuja ir kt., 2000), genetinis algoritmas su lokaliąja paieška (Lim ir kt., 2000; Merz, Freisleben, 2000), genetinis algoritmas, panaudojant suliejimo kryžminimą (Drezner, 2003), pagerintas genetinis algoritmas (Misevicius, 2004), lygiagretus genetinis algoritmas (angl. *parallel genetic algorithm*) (Tosun, Dokeroglu, Cosar, 2013), memetinis algoritmas (Benlic, Hao, 2015; Thainiam, 2019), lygiagretusis memetinis algoritmas, hibridizuotas su iteratyviąja tabu paieška (angl. *parallel memetic iterated tabu search*) (Silva ir kt., 2021), kvantinis genetinis algoritmas (Chmiel, Kwiecień, 2018) bei kitos GA modifikacijos (Tang ir kt., 2006; Tosun ir kt., 2013; Tosun, 2014; Ahmed, 2015; Harris ir kt., 2015; Ahmed, 2016; Drezner, Drezner, 2019).

2.3.5 Iteratyvioji paieška (angl. *iterated search*)

Net ir panaudojus daugelio lokaliai optimalių sprendinių paieškos metodus, pvz., tabu paiešką, vis dar išlieka rimtų sunkumų, ypač sprendžiant didesnės apimties uždavinius. Tarp įvairių galimų paminėtini šie svarbesnieji tuos sunkumus sąlygojantys faktoriai: a) milžiniškas lokaliųjų optimumų sprendinių „erdvėje“ kiekis (4 pav.); b) paieškos „trajektorijų“ ciklai; c) „determinuoto chaoso“ fenomenas (Battiti, Tecchiolli, 1994).

Visų pirma, netgi ir naudojant daugelį LO surandančius algoritmus, išnagrinėjama tik mikroskopiškai maža lokaliai optimalių sprendinių dalis. Kita opi problema yra paieškos trajektorijų ciklai; ypač tai aktualu tabu paieška grindžiamiems algoritmams. Ciklai yra viena iš priežasčių, sukeliančių paieškos stagnaciją. Prie minėtų sunkumų prisideda ir paieškos chaoso fenomenas, kuomet net ir maži pradiniai sprendinių pertvarkymai vėliau atveda prie labai atsitiktinio paieškos proceso pobūdžio, kas trukdo artėti prie aukštos kokybės sprendinių.



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

Padėti eliminuoti ar bent jau sušvelninti negatyvių faktorių įtaką gali iteratyvioji (lokalioji) paieška (IP, ILP). ILP esmė slypi tame, jog galima pagerinti tradicinių LP algoritmų rezultatus pritaikant tam tikrus papildomus gautų sprendinių pertvarkymus (Baum, 1986; Martin ir kt., 1992; Martin, Otto, 1996; Schrimpf ir kt., 2000). Nesigilinant į detales, iteratyvioji lokalioji paieška gali būti traktuojama kaip „griauti ir atstatyti“ (arba diversifikavimo-intensifikavimo) principu pagrįsta paieškos strategija (Máximo, Nascimento, 2019).

Iteratyviajai lokaliajai paieškai būdingi du pagrindiniai etapai (Den Besten ir kt., 2001; Lourenco ir kt., 2002): 1) sprendinio pagerinimas (lokalioji paieška arba intensifikavimas) ir 2) sprendinio diversifikavimas (SD) (rekonstravimas, mutavimas). Tarp šių etapų įsiterpia sprendinio-kandidato atrankos rekonstravimui procedūra. ILP pradedama duoto sprendinio pagerinimu gaunant lokaliai optimalų sprendinį, tarkime $s^{\bullet}$. Gautasis sprendinys yra tam tikru mastu rekonstruojamas, gaunant naują sprendinį, pvz., $s^{\sim}$. Rekonstruotas sprendinys $s^{\sim}$ vaidina pradinio „taško“ iš naujo startuojančiai pagerinimo procedūrai vaidmenį. LP procedūra grąžina naują lokaliai optimalų sprendinį $s^{\bullet}$, šis vėl rekonstruojamas, ir t. t. Geriausias LO įsimenamas.

Yra labai įdomi analogija tarp paieškos intensifikavimo ir diversifikavimo (iš vienos pusės) ir fizikoje žinomo entropijos fenomeno (iš kitos pusės). Iš tiesų, intensifikavimas gali būti suprantamas kaip entropijos mažėjimas; kita vertus, diversifikavimas atitinka entropijos didėjimą. Ir tikrai, entropijos mažėjimas–didėjimas vyksta atvirose realiose fizinėse, materialiose sistemose. Iliustratyvus pavyzdys yra kosmoso kūnų, būtent žvaigždžių, evoliucijos procesas. Šiuo atveju žvaigždžių (kartu su planetomis, gyvybės formomis ir pan.) gimimas, formavimasis susiejamas su aiškiu entropijos mažėjimu, o žvaigždžių gyvavimo pabaiga (supernovos) yra ryškus entropijos padidėjimas.

Iteratyviosios paieškos algoritmų neigiama savybė yra ta, jog išauga kompiuterio naudojamos atminties ir centrinio procesoriaus išeikvojamo laiko resursai, lyginant su kitais algoritmais. Be to, IP algoritmai naudoja papildomus valdymo parametrus, kurių reikšmes sureguliuoti nėra taip paprasta.

Kalbant apie KP uždavinį, iteratyvioji lokalioji paieška (angl. *iterated local search*) (Stützle, 2006; Ramkumar ir kt., 2008) ir iteratyvioji tabu paieška (angl.

iterated tabu search) (Misevicius, 2012) yra efektyvesni metodai, lyginant su klasikine lokaliąja paieška, taip pat grynąja tabu paieška. Be to, iteratyviosios paieškos procedūros gali būti hibridizuojamos su kitais algoritmais, pvz., populiaciniais algoritmais (Misevicius, 2004).

2.3.6 Hierarchiniai algoritmai (angl. *hierarchical algorithms*)

Hierarchinės iteratyviosios paieškos (HIP) algoritmai yra dar vienas iteratyviųjų euristinių algoritmų išvystymo etapas. Esminė idėja yra ta, jog hierarchinius algoritmus (HA) sudaro ne vienas (kaip tai yra iteratyviuosiuose algoritmuose), bet keli (2, 3 ir daugiau) paieškos vykdymo (atlikimo) lygmenys.

Hierarchinis principas yra universalus, visapusiškas principas, pasireiškiantis tiek objektyvioje fizinėje realybėje, tiek žmonių sukurtose intelektualiose sistemose. Tai iliustruoja daug pavyzdžių. Pavyzdžiui, tai, visų pirma, galioja energijos-materijos sandarai, pradedant pačiais smulkiausiais elementais (kvarkais, elementariosiomis dalelėmis) ir baigiant stambiausiomis struktūromis (galaktikomis, galaktikų klasteriais). Hierarchiškumas gali būti matomas ir analizuojant kūnų judėjimą: galaktikos juda galaktikų klasterių atžvilgiu, žvaigždės sukasi apie galaktikos centrą, planetos – apie žvaigždes ir t. t. Intelekto sukurtoms formalioms sistemoms irgi būdingas hierarchiškumo principas. Štai imkime skaičius. Kvaternijonai turi savyje kompleksinius skaičius, kompleksiniai skaičiai apima realiuosius skaičius, realiųjų skaičių klasė yra sudaryta iš racionaliųjų skaičių ir iracionaliųjų skaičių, o racionalieji skaičiai savo ruožtu apima sveikuosius skaičius ir natūrinius skaičius. Pagaliau žmonių veikloje taip pat galime stebėti hierarchiškumą; ypač tai akivaizdu nagrinėjant valstybių, organizacijų, įstaigų, įmonių struktūrą. Kompiuterio duomenų saugojimo sistemos irgi realizuotos pagal hierarchinį principą.

Čia aprašyta tik kai kurie hierarchinių sistemų pavyzdžiai. Jų būtų galima pateikti dar daugiau, tuomet dar labiau atsiskleistų hierarchiškumo principo mastas ir universalumas. Dėl viso to logiška ir natūralu manyti, kad hierarchinis principas galioja ir euristiniuose optimizavimo algoritmuose.

Dar reikėtų pabrėžti, kad hierarchiškumo principas yra ne tik universalus, bet ir tuo pačiu paprastas. Štai algoritmų atveju šis principas iš esmės reiškia tai, kad algoritmas kaip visuma yra panašus (savi-panašus) į savo dalį; jeigu konkrečiau, aukštesnio lygmens algoritmas „naudoja“ savi-panašų žemesniojo lygmens algoritmą ir t. t. (iki nulinio lygio). Šių lygių gali būti kiek norima daug; konkretų, apibrėžtą lygmenų skaičių nustato algoritmo kūrėjas arba vartotojas. Bendruoju atveju savi-panašūs yra šie algoritmo pagrindiniai komponentai (ingredientai): žemesniojo lygmens algoritmas (algoritmo iškvietimas), sprendinio parinkimas pertvarkymui (mutavimui), sprendinio pertvarkymas.

Hierarchinio algoritmo pagrindinis veikimo principas yra tas, jog aukštesniojo lygmens algoritmas kreipiasi į žemesniojo lygmens algoritmą, pastarasis kreipiasi į dar žemesnio lygmens algoritmą, ir taip iki pačio žemiausio lygmens algoritmo. Laikoma, jog žemiausiojo lygmens algoritmas yra 0-nio hierarchinio lygio algoritmas. Toks algoritmas nebesikreipia į save panašų algoritmą, tačiau jis vis tiek kreipiasi į sprendinio pagerinimo (lokaliosios paieškos) procedūrą (kaip tai yra iteratyviosios

paieškos atveju). Ši sprendinio pagerinimo (lokaliosios paieškos) procedūra, galima įsivaizduoti, yra savotiškas hierarchinio algoritmo „branduolys“. Būtent ši procedūra ir tik ji vykdo (galimą) sprendinio pagerinimą, t. y. atlieka betarpišką paieškos intensifikavimą – koks bebūtų paieškos hierarchinių lygių skaičius.

Galimos įvairios HA modifikacijos. (Garai, Chaudhuri, 2007) straipsnyje pateiktas hierarchinis genetinis algoritmas, kurio esminė savybė yra ta, jog naudojamos kelios sprendinių populiacijos (sub-populiacijos), o genetinė paieška lygiagrečiai vykdoma vienu metu visose sub-populiacijose. (Hussin, Stützle, 2009) straipsnyje pristatytas hierarchinis iteratyviosios lokaliosios paieškos algoritmas, kuriame realizuoti du hierarchiniai iteratyviosios lokaliosios paieškos lygmenys. Šiame algoritme nenaudojama sprendinių populiacija. (Battarra ir kt., 2011) straipsnyje pasiūlytas dviejų lygmenų genetinis algoritmas (angl. *master-slave algorithm*), kurio aukštesniajame lygmenyje vykdomi genetiniai operatoriai, žemesniajame lygmenyje atliekamas populiacijos atnaujinimas. (Schaefer ir kt., 2012) straipsnyje pateiktas vėlgi hierarchinis genetinis algoritmas, kuriame realizuotas decentralizuotas populiacijos valdymo mechanizmas, o genetiniai procesai vykdomi lygiagrečiai. (Hauschild ir kt., 2012) straipsnyje pristatytas hibridinis hierarchinis GA, kur genetiniai operatoriai yra kombinuojami (apjungiami) su hierarchine lokaliąja paieška, taikoma atskiriems sprendiniamas. (Ahmed, Sun, 2018) straipsnyje pasiūlytas hierarchinis dalelių spiečiaus optimizavimo algoritmas, kuriame sprendžiamas uždavinys hierarchiškai dekomponuojamas į atskiras dalis; atskiros dalys sprendžiamos hibridizuojant hierarchinę ir iteratyviąją paiešką. (Rokbani ir kt., 2020) straipsnyje pateiktas hierarchinis skruzdėlių kolonijos optimizavimo algoritmas. Algoritmo ypatumas yra tas, kad jame hibridizuojama skruzdėlių kolonijos optimizavimas, gamtos inspiruotas jonvabalio algoritmas (angl. *firefly algorithm*) bei lokaliosios paieškos algoritmas. Vienas iš naujausių hierarchinių algoritmų yra hierarchinis hibridinis genetinis-iteratyviosios paieškos algoritmas (Misevičius ir kt., 2021), kuris buvo labai sėkmingai pritaikytas specialiam KP uždavinio atvejui – pilkųjų šablonų uždaviniui (angl. *grey pattern problem*).

Pagrindiniai HA algoritmų trūkumai yra susiję su palyginti dideliu vykdymo (konvergavimo) laiku, išaugusiu algoritminių komponentų bei valdymo parametrų kiekiu, sudėtingesne algoritmų struktūra, programinio realizavimo komplikacijomis, taip pat ilgu algoritmų konfigūracijų derinimo procesu.

Nežiūrint HA trūkumų, pabrėžtinos yra šios preliminarios, apriorinės hierarchinių algoritmų efektyvumo prielaidos: 1) didesnis (lyginant su kitais algoritmais) vykdomų paieškos iteracijų skaičius; 2) didesnė išnagrinėjamos sprendinių erdvės apimtis (tuo pačiu didesnis aptinkamų lokaliai optimalių sprendinių skaičius); 3) didesnis sprendinių įvairoviškumas (didesnė paieškos diversifikacija).

2.3.7 Kiti algoritmai

Godžiosios randomizuotos adaptyvios paieškos procedūra (angl. *greedy randomized adaptive search procedure* (GRASP)). GRASP algoritmai gali būti naudojami kaip autonominiai algoritmai (Li ir kt., 1994) arba kombinuojami su kitais algoritmais. Taip pat gali būti naudojami pradinių sprendinių generavimui.

Populiaciniai, kolektyvinį intelektą imituojantys algoritmai (angl. *swarm intelligence algorithms*). KP uždaviniui yra išbandyti ir kolektyvinio intelekto veikseną imituojantys algoritmai: skruzdėlių kolonijų algoritmai (angl. *ant colony optimization algorithms*) (Stützle, Dorigo, 1999; Gambardella ir kt., 1999; Samanta ir kt., 2019), dalelių spiečių algoritmai (angl. *particle swarm optimization algorithms*) (Hafiz, Abdennour, 2016), bičių spiečių algoritmai (angl. *artificial bee colony algorithms*) (Dokeroglu ir kt., 2019), agentų rinkiniu (spiečiumi) pagrįstas optimizavimo metodas (angl. *multi-agent based optimization method*) (Sghir ir kt., 2018), dažnų pėdsakų paieškos algoritmas (angl. *frequent pattern-based search algorithm*) (Zhou ir kt., 2020).

Nemažą dalį algoritmų KP uždaviniui sudaro įvairūs kombinuotieji (hibridiniai) algoritmai: hibridiniai genetiniai algoritmai (HGA) (Drezner, 2003; Misevicius, 2004; Rodriguez ir kt., 2004; Xu ir kt., 2006; Misevičius, Rubliauskas, 2009; Drezner, Misevičius, 2013; Benlic, Hao, 2015; Lalla-Ruiz, Expósito-Izquierdo ir kt., 2016; Özçetin, Öztürk, 2016; Ahmed, 2018; Drezner, Drezner, 2020), hibridinis skruzdėlių kolonijų algoritmas (Stützle, Dorigo, 1999; Gambardella ir kt., 1999), hibridinis evoliucinis algoritmas (Baldé ir kt., 2020), hibridinis dalelių spiečių algoritmas (Helal, Abdelbar, 2014), hibridinis mokymo-apsimokymo algoritmas (angl. *teaching-learning-based optimization algorithm*) (Dokeroglu, 2015), hibridinis „didžiojo potvynio“ ir tabu paieškos algoritmas (angl. *great deluge and tabu search*) (Acan, Ünveren, 2015), daugkartinio vykdymo hiper-euristinis algoritmas (angl. *multistart hyper-heuristic algorithm*) (Dokeroglu, Cosar, 2016), randomizuoto dekomponavimo (angl. *randomized decomposition*) algoritmas (Mihic irk t., 2018), „bio-geografijos“ algoritmas (angl. *biogeography-based optimization algorithm*) (Lim ir kt., 2016), „didžiojo sprogimo-didžiojo susitraukimo“ (angl. *Big Bang-Big Crunch*) algoritmas (Qawqzeh ir kt., 2020), vandens tekėjimo imitavimo algoritmas (angl. *water flow algorithm*) (Ng, Tran, 2019), gėlių apvaisinimo algoritmas (angl. *flower pollination algorithm*) (Abdel-Baset ir kt., 2017), paukščių migravimo imitavimo algoritmas (angl. *migrating birds optimization algorithm*) (Duman ir kt., 2012), gegutės elgsenos imitavimo algoritmas (angl. *cuckoo search algorithm*) (Ismail ir kt., 2017), banginių elgsenos imitavimo ir tabu paieškos hibridinis algoritmas (angl. *whale algorithm with tabu search*) (Abdel-Basset ir kt., 2018), pingvinų elgsenos imitavimo algoritmas (angl. *penguins search optimization algorithm*) (Mzili ir kt., 2017), kiti euristiniai algoritmai (Sun ir kt., 2014; Szwed ir kt., 2015; Cárdenas ir kt., 2017; Chmiel, 2019; Drezner 2019; Kiliç, Yüzgeç, 2019; Dantas, Pozo, 2020; Zamani, Amirghasemi, 2020).

Pabrėžtina, kad nemažai optimizavimo specialistų gana kritiškai pasisako dėl kai kurių pačių naujausių (meta)euristinių algoritmų – visų pirma dėl to, jog yra stengiamasi manipuliuoti (galima sakyti „žongliruoti“) skambiomis metaforomis, vaizdingomis analogijomis ir sulyginimais; tuo tarpu algoritmų fundamentinis naujumas ir tikrasis idėjų originalumas yra labai abejotinas. Visa tai galioja daugeliui metaforomis besiremiančių algoritmų (angl. *metaphor-based algorithms*) – pradedant „didžiojo sprogimo“ (angl. *Bing-Bang*), baigiant „multivisatos“ (angl. *Multiverse*) algoritmais.

2.4 Tikslieji algoritmai kvadratinio paskirstymo uždaviniui

Yra žinoma, kad tikslieji algoritmai garantuoja uždavinio optimalaus sprendinio radimą. Tačiau, bendrai paėmus, kaip ir daugeliui kombinatorinio optimizavimo uždavinių, taip ir KP uždaviniui, tiksliųjų algoritmų vykdymo laikas auga eksponentiškai, didėjant uždavinio apimčiai. Nemažai tiksliųjų algoritmų KP uždaviniui pagrįsti šakų ir ribų (angl. *branch and bound*) principu (Nugent ir kt., 1968; Anstreicher, Brixius, 2001; Anstreicher ir kt., 2002), nors net ir apatinės ribos (angl. *lower bound*) apskaičiavimas, reikalingas šiems algoritmams, nėra paprasta užduotis (Gilmore, 1962; Lawler, 1973).

Kitą svarbią tiksliųjų algoritmų grupę sudaro algoritmai, besiremiantys „reformulavimo“ metodu (angl. *reformulation-linearization technique*) (Sherali, Adams, 1998; Hahn, Grant, 1998; Hahn ir kt., 2012; Gonçalves ir kt., 2017; Date, Nagi, 2019). Be šių algoritmų, yra sukurta ir kai kurių kitų algoritmų tikslių (optimalių) KP uždavinio sprendinių radimui, pvz., (Roupin, 2004; Rendl, Sotirov, 2007; Nyberg, Westerlund, 2012; Ferreira ir kt., 2018).

2.5 Apibendrinamosios pastabos ir prielaidos

Šiame (2) skyriuje apžvelgti pagrindiniai euristiniai ir tikslieji algoritmai, taikytini kvadratinio paskirstymo uždavinio sprendimui. Iš pradžių bendrai apibrėžti kombinatorinio optimizavimo uždaviniai, taip pat apibrėžtas bendrasis euristinių optimizavimo algoritmų principas bei aptartos EA savybės, akcentuojant tą faktą, jog, nežiūrint to, kad EA negarantuoja optimumo suradimo, jie leidžia gauti tinkamus vartotojui sprendinius tiek sprendinių kokybės, tiek algoritmo vykdymo laiko požiūriu. Skyriuje nušviestos populiariausių EA, orientuotų KP uždavinio sprendimui, grupės: lokalioji paieška, atkaitinimo modeliavimas, tabu paieška, genetiniai algoritmai, iteratyvioji paieška, taip pat hierarchiniai bei kai kurių kitų specifinių tipų algoritmai.

Pabrėžta, jog būtent hierarchinio tipo algoritmai yra vieni iš perspektyviausių euristinių algoritmų su svariomis potencialiomis galimybėmis.

Apsispręsta KP uždavinio sprendimui taikyti hibridiniu ir hierarchiniu principais pagrįstą algoritmą – hibridinį genetinį-hierarchinį algoritmą. Hibridinis principas pasirinktas dėl jo efektyvumo ir lankstumo kombinuojant įvairius algoritmus, o hierarchinis principas – dėl jo novatoriškumo, įvairoviškumo bei daugelio lygmenų paieškos proceso naudojimo.

3 ANALITINĖ DALIS (GENETINIS HIERARCHINIS ALGORITMAS KVADRATINIO PASKIRSTYMO UŽDAVINIUI)

3.1 Kvadratinio paskirstymo uždavinys ir jo taikymai

Kvadratinio paskirstymo (KP) uždavinys (angl. *quadratic assignment problem*) yra vienas iš sudėtingų kombinatorinio optimizavimo uždavinių (Burkard, 1984; Burkard, 1991; Burkard ir kt., 1998; Çela, 1998; Drezner, 2015). KP uždavinys aprašomas taip (Koopmans, Beckmann, 1957). Duotos sveikųjų skaičių matricos $\boldsymbol{A} = (a_{ij})_{n\times n}$ ir $\boldsymbol{B} = (b_{kl})_{n\times n}$ bei aibė Π_n, kurią sudaro natūrinių skaičių nuo 1 iki n perstatymai. Perstatymui $p = \big(p(1),\, p(2),\, \ldots,\, p(n)\big)$ galėtų atitikti, pvz., n elementų išdėstymas tam tikrose vietose – pozicijose; šiuo atveju $p(i) \in \{1,\, 2,\, \ldots,\, n\}$ žymėtų pozicijos, į kurią paskirtas i-tasis elementas, eilės numerį (žr. 5 pav.). Išspręsti KP uždavinį reiškia surasti perstatymą $p_{opt} \in \Pi_n$ ir tokį, kad $p_{opt} = \arg\min_{p\in\Pi_n} z(p)$; čia z yra KPU tikslo funkcija:

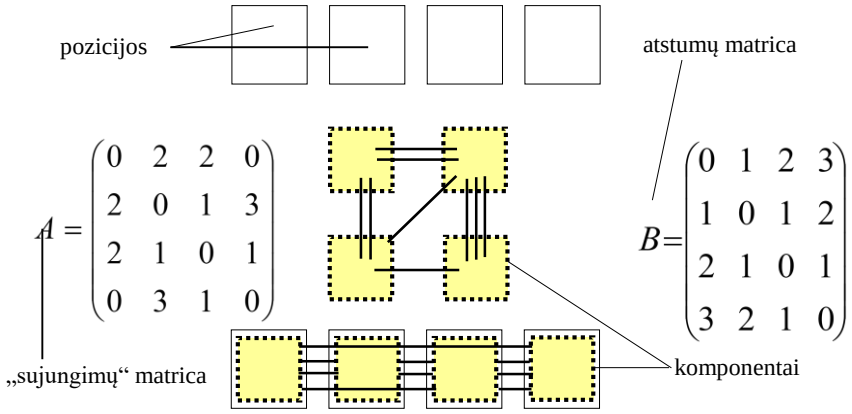
$$z(p) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} b_{p(i)p(j)}; \qquad (3)$$

p atlieka KP uždavinio sprendinių vaidmenį; n yra uždavinio apimtis.

Kvadratinio paskirstymo uždavinys pasižymi įvairiais praktiniais taikymais. Pagrindinės taikymo sritys yra šios: pastatų išdėstymo planavimas (Dickey, Hopkins, 1972; Elshafei, 1977; Hahn, Krarup, 2001); įrengimų išdėstymo planavimas (Francis, White, 1998; Drira ir kt., 2007); elektroninių komponentų (tarp jų ir mikrograndynų) projektavimas (Steinberg, 1961; Hanan, Kurtzberg, 1972; de Carvalho, Rahmann, 2006) (žr. 6 pav.); kompiuterių / telefonų klaviatūrų projektavimas (Burkard, Offermann, 1977; Dell'amico ir kt., 2009; Herthel, Subramanian, 2020); pilkųjų atspalvių formavimas kompiuterinėje grafikoje (Taillard, 1995); klasterių formavimas (Drezner, 2006); skaitmeninė analizė (Brusco, Stahl, 2000); indeksų paskirstymas (Ben-David, Malah, 2005); turbinų balansavimas (Laporte, Mercure, 1988); oro uostų terminalų projektavimas (Drezner ir kt., 2005); estafečių komandų sudėčių formavimas (Heffley, 1977); cheminių molekulinių junginių formavimas (Phillips, Rosen, 1994); vaisinių medžių sodų planavimas (Lstibůrek ir kt., 2015) ir kt. (Drezner, 2015; Abdel-Basset ir kt., 2018).

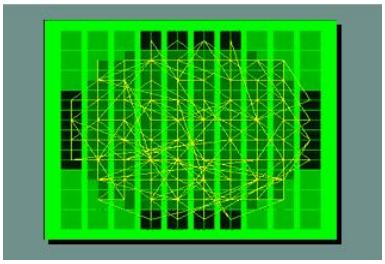
Kita vertus, pastebėta, kad KP uždavinys yra sudėtingas teorinis uždavinys. Įrodyta, jog KP uždavinys priklauso NP-sunkių optimizavimo uždavinių klasei (Sahni, Gonzalez, 1976). (Tai reiškia, kad skaičiavimų laikas, reikalingas optimumui pasiekti, yra susietas su uždavinio apimtimi n eksponentine priklausomybe, kaip minėta 1.2 sk.) KP uždavinį išspręsti tiksliai galima, kai uždavinio apimtis (n) yra labai nedidelė ($n < 30$) (https://coral.ise.lehigh.edu/data-sets/qaplib/); nors yra pavykę tiksliai išspręsti ir kai kuriuos didesnės apimties ($n = 36$ (Nyström, 1999), $n = 64$ (Drezner, 2006)) uždavinių pavyzdžius.

Dėl šios priežasties KP uždavinio sprendimui naudojami euristiniai optimizavimo algoritmai. Nors šie algoritmai, kaip minėta, negarantuoja optimalaus sprendinio suradimo, tačiau jie leidžia gauti pakankamai aukštos kokybės (artimus optimaliems) sprendinius per priimtiną skaičiavimų laiką.



Duotų „sujungimų“ ir atstumų matricų ***A*** ir ***B*** perstatymas, kuris atitinka optimalų komponentų paskirstymą į pozicijas, yra toks: (2, 3, 1, 4). Bendras sujungimų, atitinkančių šį paskirstymą, ilgis yra lygus 12.

5 pav. Paprastas komponentų išdėstymo pavyzdys



6 pav. Didelio skaičiaus elementų išdėstymo padidintos integracijos mikroschemoje pavyzdys

3.2 Bazinės formuluotės

Pradžioje pateikiame bazines formuluotes (apibrėžimus).

1. Tarkime, kad $p(u)$ $(u = 1, \dots, n)$ ir $p(v)$ $(v = 1, \dots, n,\ u \neq v)$ yra du perstatymo (sprendinio) p elementai, kurie algoritmo vykdymo eigoje gali būti sukeisti vietomis. Tuomet $p^{u,v}$ apibrėžiamas taip:

$$p^{u,v}(i) = \begin{cases} p(i), & i \neq u, v \\ p(u), & i = v \\ p(v), & i = u \end{cases}. \qquad (4)$$

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:

$$p^{u,u} = p, \tag{5}$$

$$p^{u,v} = p^{v,u}, \tag{6}$$

$$(p^{u,v})^{u,v} = p. \tag{7}$$

Taip pat bet kokiems u, v, w galioja:

$$(p^{u,v})^{v,w} = (p^{w,v})^{v,u}. \tag{8}$$

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

$$\Delta(p^{u,v}, p) = z(p^{u,v}) - z(p) = (a_{uu} - a_{vv})\left(b_{p(v)p(v)} - b_{p(u)p(u)}\right) + (a_{uv} - a_{vu})\left(b_{p(v)p(u)} - b_{p(u)p(v)}\right) + \sum_{k=1,k\neq u,v}^{n}\begin{bmatrix}(a_{uk} - a_{vk})\left(b_{p(v)p(k)} - b_{p(u)p(k)}\right) + \\ (a_{ku} - a_{kv})\left(b_{p(k)p(v)} - b_{p(k)p(u)}\right)\end{bmatrix}; \tag{9}$$

čia: a_{uu}, ..., $b_{p(v)p(v)}$, ... yra atitinkami matricų A, B elementai (žr. (3) formulę).

Tikslo funkcijos pokytį dviem „perstatymams-kaimynams“ p ir p', kai $p' = p^{u,v}$, galima apskaičiuoti ir žymiau greičiau – su sąlyga, jog yra įsimintos (išsaugotos) visos ankstesnių pokyčių reikšmės $\Delta(p^{i,j}, p)$ $(i, j = 1, \ldots, n)$. Pokyčių saugojimui pakanka dvimatės matricos, o pokyčio reikšmė gaunama, panaudojant $O(1)$ operacijų (Frieze ir kt., 1989; 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:

$$\Delta'(p^{i,j}, p) = \Delta(p^{i,j}, p) + (a_{iu} - a_{iv} + a_{jv} - a_{ju}) \times \left(b_{p(i)p(u)} - b_{p(i)p(v)} + b_{p(j)p(v)} - b_{p(j)p(u)}\right) + (a_{ui} - a_{vi} + a_{vj} - a_{uj}) \times \left(b_{p(u)p(i)} - b_{p(v)p(i)} + b_{p(v)p(j)} - b_{p(u)p(j)}\right). \tag{10}$$

Pastaba. Jeigu $i = u$ arba $i = v$, arba $j = u$, arba $j = v$, tai turi būti taikoma (9) formulė. Nors pastarosios formulės sudėtingumas yra $O(n)$, tačiau ji taikoma vos 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 (9) 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:

$$\Delta(p^{u,v}, p) = \sum_{k=1,k\neq u,v}^{n}(a'_{uk} - a'_{vk})\left(b_{p(v)p(k)} - b_{p(u)p(k)}\right); \tag{11}$$

čia $a'_{uk} = a_{uk} + a_{ku},\ u = 1, \dots, n,\ v = 1, \dots, n,\ u \neq v$. Analogiškai (10) formulė virsta tokia formule:

$$\Delta'\left(p^{i,j}, p\right) = \Delta\left(p^{i,j}, p\right) + \left(a'_{iu} - a'_{iv} + a'_{jv} - a'_{ju}\right) \times \left(b_{p(i)p(u)} - b_{p(i)p(v)} + b_{p(j)p(v)} - b_{p(j)p(u)}\right). \quad (12)$$

Jeigu $i = u(v)$ arba $j = u(v)$, tai turi būti taikoma (11) 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 $\boldsymbol{B}'' = (b''_{lrt})_{n\times n\times n}$. 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 galima apskaičiuoti labai kompaktiškomis formulėmis:

$$\Delta(p^{u,v}, p) = \sum_{k=1, k\neq u,v}^{n} a''_{uvk} b''_{p(v)k(u)p(k)}, \quad (13)$$

$$\Delta'\left(p^{i,j}, p\right) = \Delta\left(p^{i,j}, p\right) + \left(a''_{iju} - a''_{ijv}\right) \times \left(b''_{p(j)p(i)p(v)} - b''_{p(j)p(i)p(u)}\right). \quad (14)$$

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 išankstiniai praktiniai eksperimentai patvirtina prielaidą, jog galima sutaupyti iki 20 % procesorinio laiko. (Aišku, kompiuterinės atminties sunaudojimas išauga: matricų $\boldsymbol{A}''$, $\boldsymbol{B}''$ saugojimui reikia $O(n^3)$ atminties.)

3. Atstumas (Hemingo atstumas) tarp dviejų perstatymų p_1 ir p_2 apibrėžiamas taip:

$$\delta(p_1, p_2) = |\{i: p_1(i) \neq p_2(i)\}|. \quad (15)$$

Galioja tokios formulės:

$$\delta(p, p) = 0, \quad (16)$$

$$\delta(p_1, p_2) \neq 1, \quad (17)$$

$$0 \leq \delta(p_1, p_2) \leq n, \quad (18)$$

$$\delta(p_1, p_2) = \delta(p_2, p_1), \quad (19)$$

$$\delta(p, p^{u,v}) = 2. \quad (20)$$

Jei turime k skirtingų $u_1, u_2, \dots, u_k$, tai yra teisinga ši formulė:

$$\delta\left((((p^{u_1,u_2})^{u_2,u_3})^{\cdots})^{u_{k-1},u_k}\right) = \xi. \quad (21)$$

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: $p^{1,2}$, $p^{1,3}$, …, $p^{1,n}$, $p^{2,3}$, …, $p^{2,n}$, …, $p^{i-1,n}$, $p^{i,i+1}$, …, $p^{i,n}$, …, $p^{n-1,n}$. Tokiu būdu aplinkos Θ_2 dydis $|\Theta_2|$ yra lygus $\left(n(n-1)\right)/2$. Tiek žingsnių reikia atlikti, norint pilnai išnagrinėti esamo sprendinio p aplinką $\Theta_2(p)$.

Taigi, šios aplinkos išnagrinėjimui pakanka $O(n^2)$ operacijų. Gali būti apibrėžiamos ir kitokios aplinkos. Sprendinių aplinkų visuma sudaro sprendinių paieškos aibę (paieškos erdvę). Perėjimą iš sprendinio p į aplinkos Θ_2 sprendinį p' galima aprašyti, panaudojant formalų dviejų perstatymo elementų sukeitimo operatorių $\phi(p,u,v)$: $\Pi_n \times \mathrm{N} \times \mathrm{N} \to \Pi_n$, kuris sukeičia perstatymo u-tąjį ir v-tąjį elementus vietomis, t. y. $\phi(p,u,v) = p^{u,v}$.

3.3 Genetinio hierarchinio algoritmo variantai kvadratinio paskirstymo uždaviniui

Kvadratinio paskirstymo uždaviniui spręsti yra naudojamas genetinis hierarchinis algoritmas (GHA) ir įvairios jo modifikacijos (variantai). Pagrindinės sudaryto algoritmo sudėtinės dalys (komponentai) yra tokios:

- pradinės populiacijos sudarymas;
- sprendinių-tėvų atrinkimas;
- sprendinių kryžminimas;
- sprendinių-palikuonių pagerinimas panaudojant hierarchinės iteratyviosios tabu paieškos (HITP) procedūrą;
- populiacijos atnaujinimas;
- populiacijos perkrovimas (algoritmo stagnavimo atveju).

Pastaba. Sprendinių mutavimas realizuotas pačioje HITP procedūroje.

Svarbiausios genetinio hierarchinio algoritmo savybės yra šios:

1. Algoritmo gaunamų palikuonių-sprendinių pagerinimui naudojamas hierarchinės iteratyviosios tabu paieškos (HITP) principu besiremiantis algoritmas. Pats genetinis algoritmas pasižymi originalia struktūra ir laikytinas genetinės paieškos ir HITP hibridu. Tuo tarpu, HITP yra pagrįsta kelių lygmenų iteratyviąja tabu paieška (ITP).
2. Pačiame HITP algoritme kombinuojama tabu paieška (paieškos intensifikavimui) ir mutavimo procedūra sprendinių pertvarkymui (paieškos diversifikavimui).
3. Genetinis algoritmas naudoja labai aukštos kokybės sprendinių populiaciją. Norint gauti kuo geresnę pradinę populiaciją, populiacijos pradinio formavimo metu naudojamas HITP algoritmas su padidintu iteracijų skaičiumi. Pradinė populiacija gali būti sukuriama, panaudojant ir patį genetinį algoritmą (pavaldųjį genetinį algoritmą (angl. *slave genetic algorithm*), kurio struktūra yra iš esmės tokia pati, kaip ir pagrindinio genetinio algoritmo). Pradinės populiacijos dydis taip pat yra padidintas; po populiacijos suformavimo atrenkami tik geriausi sprendiniai, kurie naudojami tolesnėje algoritmo eigoje.
4. Sprendinių kryžminimui naudojamos pasirinktinai suliejimo kryžminimo procedūra ir originali universaliojo kryžminimo procedūra (angl. *cohesive crossover*, *universal crossover*).

5. Populiacijos atnaujinimui panaudojama modifikuota taisyklė, kurioje įvertinama ne tik sprendinių kokybė, bet ir atstumai tarp sprendinių. Taip yra užtikrinama populiacijos individų įvairovė GA vykdymo eigoje.
6. Genetinio algoritmo stagnacijos eliminavimui naudojamas populiacijos perkrovimas (angl. *population restart*), t. y. visiems populiacijos sprendiniams taikoma mutavimo procedūra, siekiant gauti naują diversifikuotą populiaciją. (Stagnavimo identifikavimui pasinaudojama „tuščios eigos“ generacijų kriterijumi.)
7. GA algoritmo stabdymo sąlyga yra algoritmo iteracijų (t. y. generacijų) skaičius.

Formalizuotas sudaryto genetinio hierarchinio algoritmo aprašas (pseudokodas) yra pateiktas 7 pav. [1]

[1] Visus genetinio hierarchinio algoritmo programos išeities tekstus C# programavimo kalboje galima rasti internete adresu: https://www.personalas.ktu.lt/~alfmise/.

```
procedure Genetinis_hierarchinis_algoritmas_KP_uždaviniui;
//pradiniai duomenys: n – uždavinio apimtis, A, B – duomenų matricos,
//   PD – populiacijos dydis, N_gen – generacijų skaičius,
//   PPGV – pradinės populiacijos generavimo variantas,
//   TAV – sprendinių-tėvų atrinkimo variantas, KV – kryžminimo variantas,
//   MV – mutavimo variantas, PAV – populiacijos atnaujinimo variantas,
//   PPV – populiacijos perkrovimo variantas, σ – sprendinių-tėvų atrinkimo koeficientas,
//   ε – minimalaus atstumo slenkstis, ω – populiacijos stagnacijos koeficientas
//rezultatai: p* – geriausias rastas sprendinys (perstatymas)
01 begin
02  sukurti atsitiktiniu būdu pradinę populiaciją P ⊂ Π_n (|P| = PD);
03   if PPGV = 1 then
04       optimizuoti kiekvieną populiacijos P sprendinį,
05       panaudojant hierarchinės tabu paieškos algoritmą;
06    if PPGV = 2 then
07       optimizuoti populiaciją P, panaudojant pavaldųjį genetinį
08       algoritmą;
09    p* := arg min_{p∈P} z(p);  //įsimenamas geriausias pradinės populiacijos sprendinys
10    for i := 1 to N_gen do begin //pagrindinis ciklas
11      surūšiuoti populiacijos P narius tikslo funkcijos didėjimo
12      tvarka;
13      atrinkti „sprendinius tėvus“ p′, p″ ∈ P, atsižvelgiant į
14      tėvų atrinkimo variantą TAV ir atrinkimo koeficientą σ;
15      //palikuonio sukūrimas ir pagerinimas
16      atlikti „sprendinių tėvų“ kryžminimą, atsižvelgiant į
17      kryžminimo variantą KV, gauti palikuonį p‴;
18      optimizuoti gautą palikuonį p‴, panaudojant hierarchinės
19      tabu paieškos algoritmą;
20      suformuoti naują populiaciją P (|P| = PD) iš sąjungos
21      P ∪ {p‴}, atsižvelgiant į minimalų atstumą ε ir
22      populiacijos atnaujinimo variantą PAV;
23      if z(p‴) < z(p*) then p* := p‴;  //įsimenamas geriausias
24      rastas sprendinys
25      apskaičiuoti populiacijos stagnacijos kriterijų PS
26      (jeigu populiacija nesikeičia L = ωN_gen generacijose,
27      tai PS = TRUE; kitu atveju PS = FALSE);
28      if PS = TRUE then begin
29         atlikti populiacijos perkrovimą, atsižvelgiant į
30         populiacijos perkrovimo variantą PPV;
31         if z(arg min_{p∈P} z(p)) < z(p*) then p* := arg min_{p∈P} z(p)
32      endif
33    endfor
34  end.
```

Pastaba. Atlikus populiacijos perkrovimą, visi populiacijos sprendiniai yra pagerinami panaudojant hierarchinės iteratyviosios TP algoritmą.

7 pav. Genetinio hierarchinio algoritmo KP uždaviniui aprašymas (pseudokodas)

3.4 Genetinio algoritmo kryžminimo procedūros

Kryžminimo procedūros genetiniuose algoritmuose atlieka labai svarbų, jei ne svarbiausią vaidmenį. Standartinis dviejų sprendinių-tėvų kryžminimo procesas (Herrera ir kt., 2003; Umbarkar, Sheth, 2015; Pavai, Geetha, 2017) perstatymų atveju formalizuotai apibrėžiamas, panaudojant operatorių $\psi : \Pi_n \times \Pi_n \to \Pi_n$ ir tokį, jog:

$$p^\circ = \psi(p', p'') \neq p' \vee p^\circ = \psi(p', p'') \neq p'', \text{ jeigu } p' \neq p''; \qquad (22)$$

čia p', p'', p° yra sprendiniai-perstatymai (p', p'' žymi sprendinius-tėvus, p° – sprendinį-palikuonį). (Teoriškai vaikas gali sutapti su vienu iš tėvų, pvz., jeigu tėvai yra labai panašūs.) Perstatymas p gali būti asocijuojamas su genetikoje įprasta chromosoma; tuomet perstatymo elementą $p(i)$ galima traktuoti kaip atskirą geną (tiksliau, to geno turinį – alelę), esantį i-oje chromosomos pozicijoje (čia n galima laikyti chromosomos ilgiu).

Kryžminimo operatorius turi užtikrinti, jog būsimo palikuonio chromosoma būtinai paveldės tuos genus, kurie yra bendri abiem tėvų chromosomoms, t. y.:

$$p'(i) = p''(i) \Rightarrow p^\circ(i) = p'(i) = p''(i), i = 1, 2, \ldots, n; \qquad (23)$$

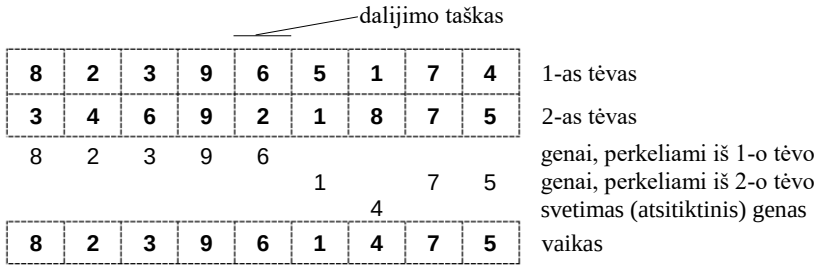
čia ir vėl p', p'', p° žymi atitinkamai sprendinius-tėvus ir sprendinį-palikuonį.

Genetiniame hierarchiniame algoritme realizuotų konkrečių kryžminimo procedūrų veikimo aprašymai pateikiami žemiau.

3.4.1 Dalijimo taško kryžminimas

3.4.1.1 Vieno taško kryžminimas

Vieno taško kryžminimo procedūra (8 pav.) (angl. *one point crossover*) (Goldberg, 1989) laikytina viena iš klasikinių procedūrų, skirtų genetiniams algoritmams. Ši procedūra nėra sudėtinga. Jos esminė idėja yra tokia. Viename iš sprendinių (chromosomoje) parenkamas dalijimo taškas. Taško vieta *x* nustatoma, generuojant (pseudo-)atsitiktinį skaičių iš intervalo $[1, n-1]$ (n – chromosomos ilgis). Sprendinys-palikuonis gaunamas, klonuojant vieną dalį genų iš pirmojo tėvo (genai nuo 1-ojo iki *x*-ojo imtinai), likusi genų dalis kopijuojama iš antrojo tėvo. Perkeliant genus iš antrojo tėvo, turi būti užtikrinama, jog gautasis sprendinys-palikuonis išliktų perstatymu. Vieno taško kryžminimo pagrindu galima sukonstruoti 2, 3 (ir t. t.) taškų kryžminimo procedūras.



8 pav. Vieno taško kryžminimo procedūros veikimo iliustracija

Vieno taško kryžminimo neigiamas aspektas yra tas, jog gaunamos palikuonio chromosomos viena dalis visuomet yra tendencingai labiau „tvarkinga“, negu kita (palyginkite, pvz., pavaizduotas dalis „82396“ ir „1475“: pirmoje dalyje nėra jokių svetimų genų, antroje dalyje – yra). Šio tendencingo nukrypimo galima išvengti, tik vos pakoregavus originalų algoritmą. Jeigu konkrečiau, modifikuotoje vieno taško kryžminimo procedūroje palikuonis gaunamas su tikimybe, lygia $\frac{1}{2}$, kopijuojant iš pirmojo tėvo arba genus nuo 1-ojo iki x-ojo, arba genus nuo x-ojo iki n-ojo (n – chromosomos dydis); likusi dalis kopijuojama iš antrojo tėvo.

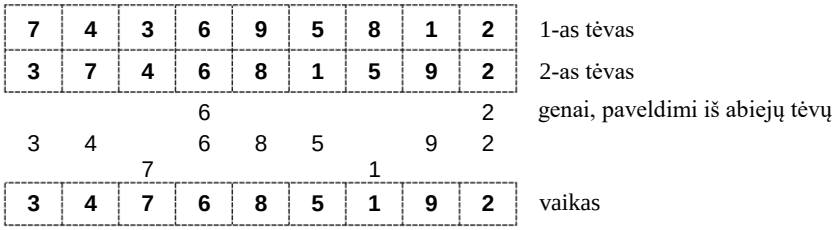
3.4.1.2 Dviejų taškų kryžminimas

Dviejų taškų kryžminimo procedūra veikia panašiai kaip ir vieno taško kryžminimo procedūra, tik vietoje vieno dalijimo taško x naudojami du taškai x_1 ir x_2 ($1 \leq x_1 \leq x_2 \leq n$).

3.4.1.3 Tolygiojo pasiskirstymo kryžminimas

Tolygiojo pasiskirstymo kryžminimo algoritmo (angl. *uniform crossover*) (Syswerda, 1989) veikimo principas taip pat yra paprastas. Iš pradžių, bendri abiejų tėvų chromosomų genai (vienodi elementai tose pačiose perstatymų pozicijose) yra kopijuojami į atitinkamą palikuonio chromosomos poziciją. Po to nuoseklia tvarka tikrinamos neužpildytos palikuonio chromosomos pozicijos: kiekvienai iš jų priskiriama geno reikšmė, parenkant šią iš vienos iš tėvų chromosomų su tikimybe, lygia $\frac{1}{2}$; kitaip tariant, $p^{\circ}(i) = \begin{cases} p'(i), & \zeta < \frac{1}{2} \\ p''(i), & \text{kitu atveju} \end{cases}$; čia ζ yra tolygiojo pasiskirstymo (pseudo-)atsitiktinis skaičius iš intervalo $[0, 1]$. Priskiriama reikšmė, aišku, neturi būti priskirta anksčiau. Jeigu po antrojo žingsnio palikuonio chromosomoje lieka tuščių pozicijų, tai šios užpildomos atsitiktiniu būdu, bet taip, jog gautasis sprendinys išliktų perstatymu. Tuščias pozicijas būtų galima užpildyti ir kitaip, pvz., taip, jog būtų suformuojamas sprendinys, kuriam tikslo funkcijos reikšmė yra geriausia. Tokia kryžminimo procedūra būtų euristinio pobūdžio, kadangi šiuo atveju atsižvelgiama į sprendžiamo uždavinio specifiką ir gaunamo palikuonio kokybę.

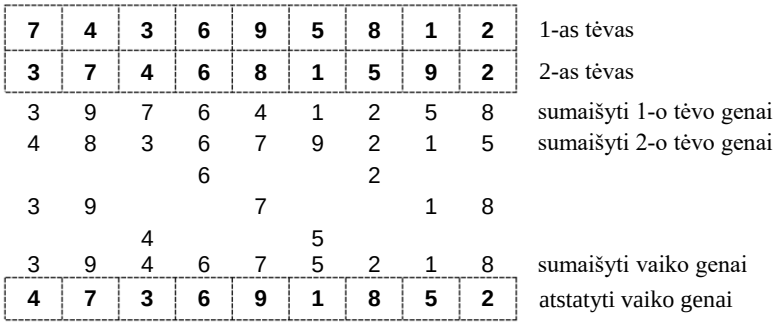
Tolygiojo pasiskirstymo kryžminimo procedūrą (9 pav.) galima modifikuoti taip, kad palikuonio chromosomos reikšmių priskyrimo tikimybė būtų ne $\frac{1}{2}$, bet koks nors realus skaičius iš intervalo $[\varepsilon, 1-\varepsilon]$, $\varepsilon > 0$. Pvz., tikimybė galėtų būti lygi $\frac{3}{4}$ – tuomet palikuonis paveldėtų daugiau genų iš kurio nors jo tėvo. Tokią procedūrą galima vadinti kvazi-tolygiojo pasiskirstymo kryžminimo procedūra.



9 pav. Tolygiojo pasiskirstymo kryžminimo procedūros veikimo iliustracija

3.4.2 Randomizuotas kryžminimas (sumaišymo kryžminimas)

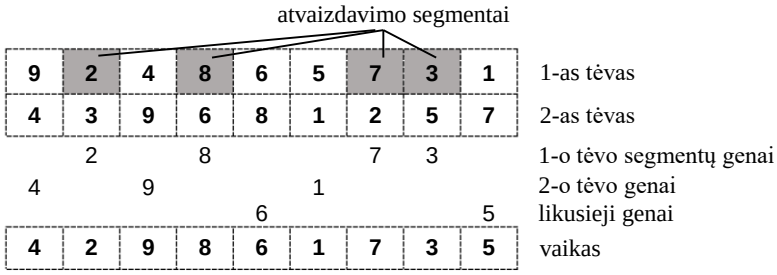
Randomizuotą kryžminimą gausime, jeigu iš pradžių pagal kurią nors taisyklę sumaišysime abiejų tėvų chromosomų genus (abiem tėvams – ta pati taisyklė) (Caruana ir kt., 1989). Tada atliekamas tolygiojo (arba) kvazi-tolygiojo pasiskirstymo kryžminimas bei sumaišytų genų atstatymas pagal pradinę taisyklę (10 pav.). Kryžminimo randomizavimo būdų yra ir daugiau.



10 pav. Randomizuoto (sumaišymo) kryžminimo procedūros veikimo iliustracija

3.4.3 Dalinio atvaizdavimo kryžminimas

Dalinio atvaizdavimo kryžminimas (angl. *partially-mapped crossover*) (Goldberg, Lingle, 1985) yra tam tikras daugelio taškų (*k*-taškų) ir randomizuoto kryžminimo kombinuotas variantas. Pradinė šio kryžminimo versija buvo skirta garsiajam komivojažieriaus uždaviniui, bet šio kryžminimo idėja gali būti adaptuota ir kitiems optimizavimo uždaviniams. Kryžminimo principo pagrindas yra atvaizdavimo sritys, kitaip tariant, chromosomos segmentai tarp *k* atvaizdavimo taškų. Į vaiko chromosomą iš pradžių perkeliami vieno pasirinkto tėvo chromosomos segmentai, po to – kito tėvo genai, galiausiai užpildomi likusieji tušti lokusai (žr. 11 pav.). Perkėlimo (t. y. segmentų nagrinėjimo) pati tvarka gali būti determinuota arba atsitiktinė. Pastebėtina, kad jeigu $k = n$, tai dalinio atvaizdavimo kryžminimas tampa labai primenantis tolygiojo pasiskirstymo kryžminimą.



11 pav. Dalinio atvaizdavimo kryžminimo procedūros veikimo iliustracija

3.4.4 Sukeitimais pagrįstas kryžminimas

3.4.4.1 Sukeitimų kryžminimas – I

Čia pristatoma kryžminimo procedūra skiriasi nuo prieš tai aptartų tuo, kad ji yra specifinio pobūdžio ir orientuota, visų pirma, perstatymus naudojantiems optimizavimo uždaviniams. Tai yra sukeitimų procedūra (angl. *swap path crossover*) (Glover, 1994). Šios procedūros ypatybė ta, jog genai ne perkeliami iš tėvų chromosomų į vaiko chromosomą, o tiesiog sukeičiami tėvų chromosomose. Tegul (p', p'') yra sprendinių-tėvų pora. Tuomet pradedama nuo kurios nors pasirinktos pozicijos (t. y. lokuso), po to tėvų poros genai skenuojami iš kairės į dešinę tol, kol atliekamas tam tikras nustatytas sukeitimų skaičius s_{suk} ($s_{suk} \leq n$). Jeigu duotajame chromosomos lokuse genų reikšmės yra tos pačios abiem tėvams, tai pereinama nagrinėti kitą geną; priešingu atveju, atliekamas porinis genų sukeitimas tėvų chromosomose. Sukeitimas atliekamas ir pirmojo tėvo chromosomoje p', ir antrojo tėvo chromosomoje p''. Būtent: jeigu esamas lokuso indeksas yra i ir $a = p'(i),\ b = p''(i)$, tai egzistuos lokusas j ir toks, jog $b = p'(j),\ a = p''(j)$; tuomet po sukeitimo $p'(i)$ tampa lygus b, $p''(i)$ – lygus a, be to, $p'(j) = a,\ p''(j) = b$. Taip po sukeitimo gaunami atitinkamai sprendiniai p''' ir p''''. Tuomet kitoje iteracijoje nagrinėjami poros (p''', p'''') genai, pradedant lokusu $i + 1$, ir t. t. (žr. 12 pav.).

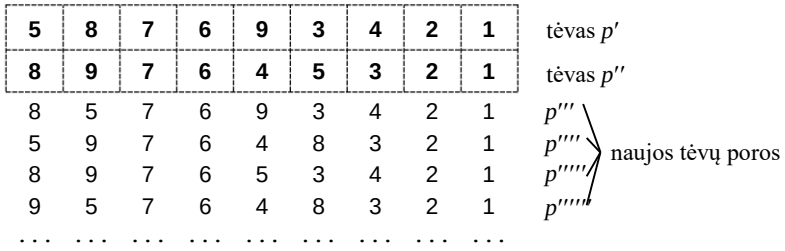
3.4.4.2 Sukeitimų kryžminimas – II (euristinis sukeitimų kryžminimas)

Galima modifikuoti sukeitimų procedūrą taip, kad genų sukeitimų ciklo vykdymo metu būtų įsimenamas tas gautas individas (sprendinys), kuriam tikslo funkcijos reikšmė yra geriausia. Tokiu būdu gaunama euristinė sukeitimų procedūra, kurioje įvertinama palikuonio kokybė.

3.4.4.3 Sukeitimų kryžminimas – III (modifikuotas euristinis sukeitimų kryžminimas)

Racionali ir perspektyvi yra modifikuota euristinė sukeitimų procedūra, kurioje sukeitimų ciklo vykdymo eigoje sukeičiami tik tie chromosomos genai, kurie gautam sprendiniui pagerina tikslo funkcijos reikšmę. (Genų sukeitimų kiekis nėra ribojamas. Paprastai į tėvų chromosomas „negrįžtama“.) Taip sukeitimų procedūroje dinamiškai

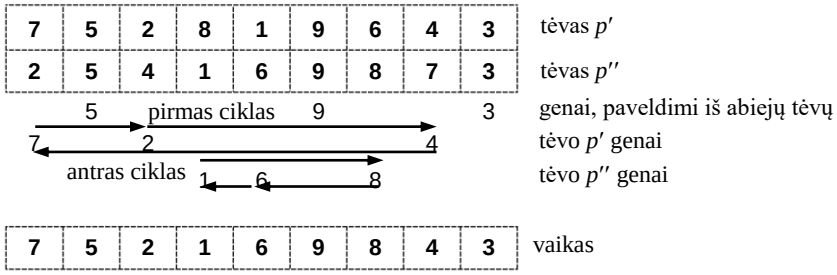
įvertinamas sudaromo sprendinio gerumas, ir, tikėtina, kad dėl šios priežasties tokia sukeitimų procedūra galėtų būti pranašesnė, lyginant, pvz., su dalijimo taško procedūra, kuri yra universalaus pobūdžio (nepriklauso nuo sprendžiamo uždavinio).



12 pav. Sukeitimų kryžminimo procedūros fragmento iliustracija

3.4.5 Ciklinis kryžminimas

Ciklinio kryžminimo procedūra (angl. *cycle crossover*) (Oliver ir kt., 1987) kiek primena sukeitimų procedūrą, t. y. kryžminimas remiasi tėvų chromosomų genų sukeitimais, atliekamais atskiroje chromosomoje. Esminė šio tipo kryžminimo skiriamoji ypatybė yra ta, jog garantuojamas tėvų chromosomose esančios informacijos perdavimas palikuoniui be jokių iškraipymų (svetimų genų). Trumpai tariant, ciklinis kryžminimas įgalina sukurti tokį palikuonį, kurio genai būtinai yra arba iš pirmojo, arba iš antrojo tėvo. Detalesnį ciklinio kryžminimo algoritmo aprašą galima rasti (Misevičius, 2008) straipsnyje. Negatyvi ciklinio kryžminimo savybė yra ta, jog palikuonis gali būti genetiškai labai artimas (arba net identiškas) vienam iš tėvų. Pvz., jeigu pateiktame pavyzdyje antrojo tėvo chromosomą nežymiai pakeistume į „254169783“, tai sugeneruota ciklinio kryžminimo metu palikuonio chromosoma („752819643“) būtų visiškai identiška pirmojo tėvo chromosomai (žr. 13 pav.).



13 pav. Ciklinio kryžminimo procedūros veikimo iliustracija

3.4.6 Suliejimo (surišimo) kryžminimas

Suliejimo (surišanti) kryžminimo procedūra pasiūlyta Dreznerio 2003 m. kaip perspektyvus būdas rekombinuoti individų genus genetiniame algoritme sprendžiant

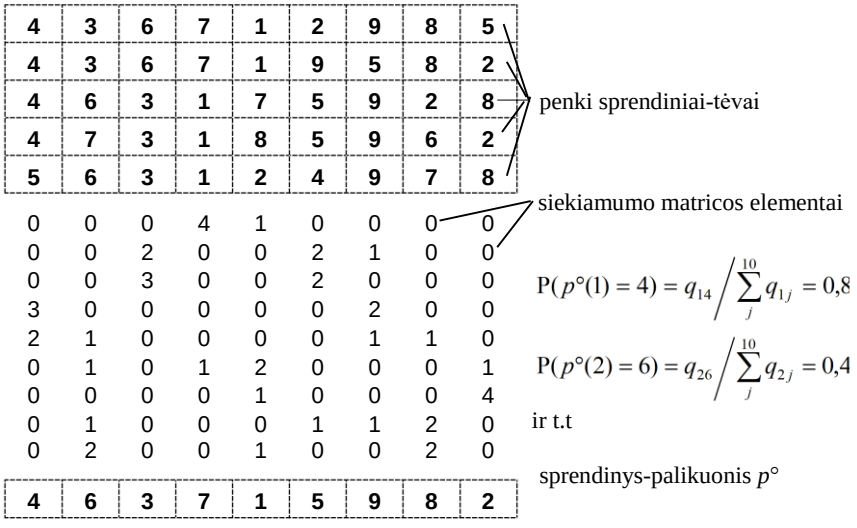
kvadratinio paskirstymo uždavinį. Nustatant tėvų rekombinuojamus genus, yra atsižvelgiama į genų vertingumą, kuris apskaičiuojamas pagal uždavinio duomenis. Iš kelių gaunamų genų (re)kombinacijų išrenkama ta, kuriai tikslo funkcijos reikšmė yra geriausia. Suliejimo procedūra taip pat laikytina euristine, specializuota procedūra, suderinta konkrečiam uždaviniui. Suliejimo procedūra nuodugniai aprašyta (Drezner, 2003) straipsnyje.

3.4.7 Daugelio tėvų kryžminimas

Galima sudaryti ir tokią kryžminimo procedūrą, kuri būtų pagrįsta ne dviejų, bet kelių tėvų panaudojimu (Eiben ir kt., 1994). Šiuo atveju palikuonis paveldi informaciją iš daugelio tėvų. Naudojant daugelio tėvų kryžminimo procedūrą (angl. *multi-parent crossover*), palikuonio chromosomos $p°$ i-ajam geno lokusui yra priskiriama kuri nors reikšmė (j) iš tėvų genų su tikimybe $P(p°(i) = j)$ – su sąlyga, jog j nėra lygus nė vienai kitai anksčiau priskirtai reikšmei. Tikimybė $P(p°(i) = j)$ apskaičiuojama pagal formulę:

$$P(p°(i) = j) = \frac{q_{i,j}}{\sum_j^n q_{i,j}}; \tag{24}$$

čia $P(p°(i) = j)$ žymi tikimybę, kad $p°(i) = j$, $q_{i,j}$ yra siekiamumo matricos $Q = (q_{i,j})_{n\times n}$ elementas; šios matricos elementas q_{ij} yra skaičius kartų to, kad i-ojo lokuso reikšmė lygi j tėvų chromosomose. Suprantama, kad $q_{ij} \leq t$, čia t – tėvų skaičius. Jeigu egzistuoja kelios reikšmės $(j_1, j_2, \dots)$ su ta pačia tikimybe, tai lokusui reikšmė priskiriama atsitiktiniu būdu (svarbu tik, kad ta reikšmė nebūtų priskirta anksčiau). Vadovaujantis tokiu principu, suformuojami visi palikuonio $p°$ genai $p°(i)$, $i = 1, \dots, n$ (14 pav., taip pat žr. Misevičius, 2006).



14 pav. Daugelio tėvų kryžminimo procedūros veikimo iliustracija

3.4.8 Universalusis kryžminimas

Siūlomas naujas, originalus kryžminimo algoritmas, kuris sąlyginai pavadintas „universalusis kryžminimas“. Terminas „universalusis“ pasirinktas, norint akcentuoti siūlomo kryžminimo principo santykinį daugiapusiškumą, galimybę jį lanksčiai valdyti pagal konkrečius vartotojo / tyrėjo poreikius.

Šis kryžminimas yra grindžiamas bitų „kaukės“ (angl. *mask*) panaudojimu, o kryžminimo algoritmas valdomas keturiais parametrais $\alpha,\ \beta,\ \gamma,\ \chi$. Kaukės ilgis yra lygus α, čia α yra (pseudo-)atsitiktinis sveikas skaičius iš intervalo $[\varepsilon_1, n]$, n yra chromosomos ilgis, $\varepsilon_1 = \lfloor h{\cdot}n \rfloor$, h – vartotojo pasirenkamas parametras, kurio reikšmė yra artima 1, pvz., 0,9. Kaukę sudaro dvejetainiai skaičiai 0 ir 1, kur 1 reiškia, jog turi būti klonuojamas pirmojo tėvo chromosomos atitinkamas genas, o 0 rodo, jog klonuojamas antrojo tėvo genas. Pagrindinis tikslas yra sukurti kuo įvairiapusiškesnę atsitiktinę bitų kaukę. Šios kaukės atsitiktinumo laipsnis gali būti kontroliuojamas parametrais $\beta,\ \gamma$. Parametras β ($\beta \in [\varepsilon_2, \varepsilon_3], 0 < \varepsilon_2 < \varepsilon_3 < 1$) leidžia reguliuoti 0 ir 1 kiekį bitų kaukėje: kuo didesnė β reikšmė, tuo daugiau vienetų yra generuojama bitų eilutėje, ir atvirkščiai. Bitų atsitiktinis išsidėstymas bitų kaukėje valdomas parametru γ. Pats bitų generavimas atliekamas, panaudojant indeterminuotą minmakso rūšiavimo algoritmą. Randomizuota bitų kombinacija gaunama, atsitiktiniu momentu nutraukiant šį algoritmą. Nutraukimo momentą apibrėžia skaičius z, čia $z = \gamma{\cdot}w$, kur γ yra (pseudo-)atsitiktinis skaičius iš intervalo $[0, 1]$, w – maksimalus indeterminuoto minmakso algoritmo iteracijų skaičius, reikalingas pilnai surūšiuoti kaukės bitus. (Pvz., rūšiuojama bitų kaukė „000001111“, rūšiavimas nutraukiamas momentu $\gamma = 0{,}9$, ir taip gaunama atsitiktinė kaukė „101100010“.) Turint atsitiktinę bitų kaukę, nusprendžiama, kurie ir kokių tėvų chromosomų genai bus perkeliami į vaiko chromosomą. Pradinė perkeliamų genų pozicija χ sugeneruojama atsitiktinai (t. y. $\chi \in [1, n]$). Paskutiniame universaliojo kryžminimo procedūros etape likę neužpildyti palikuonio chromosomos lokusai užpildomi atsitiktiniu būdu tais genais, kurių dar nėra palikuonio chromosomoje.

Iliustracinis universaliojo kryžminimo procedūros pavyzdys (kai $\alpha = 9,\ \beta = \frac{4}{9}$, $\gamma = 0{,}9$, $\chi = 1$) pateiktas 15 pav.

9	**7**	**2**	**1**	**3**	**6**	**5**	**4**	**8**	1-as tėvas
1	**6**	**3**	**4**	**8**	**5**	**9**	**2**	**7**	2-as tėvas
1	0	1	1	0	0	0	1	0	atsitiktinė bitų kaukė
9		2	1				4		genai, perkeliami iš 1-o tėvo
	6			8	5			7	genai, perkeliami iš 2-o tėvo
						3			atsitiktinis genas
9	**6**	**2**	**1**	**8**	**5**	**3**	**4**	**7**	palikuonis

15 pav. Universaliojo kryžminimo procedūros veikimo iliustracija

3.5 Sprendinių pagerinimas panaudojant tabu paiešką

3.5.1 Tabu paieškos algoritmas

Tabu paieškos procesas formaliai primena tą procesą, kuris aprašytas (Misevičius ir kt., 2021) straipsnyje. TP proceso eigoje analizuojama esamojo sprendinio p aplinka $\Theta_2(p)$, 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. Siekiant eliminuoti paieškos ciklus, atvirkštinis perėjimas, taipogi grįžimas į kitus neseniai aplankytus sprendinius turi būti uždraudžiamas apibrėžtam laikotarpiui. Draudimai saugomi tabu atmintyje, t. y. sąraše T, įsimenant paskutinius nagrinėtus sprendinius. Tabu sąrašas T realizuotas kaip dvimatė $n \times n$ dydžio matrica. Šiuo atveju matricos elementas t_{ij} saugo esamos iteracijos numerį plius uždraudimo periodą (angl. *tabu tenure*) h; tokiu būdu ši reikšmė nurodo, nuo kurios iteracijos vėl galima sukeisti perstatymo i-tąjį ir j-tąjį narius (Taillard, 1991). Parametro h reikšmė 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 ($\alpha = 0{,}02$). (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 tabu paieškos procedūros eigoje rastą geriausią sprendinį. Tabu sąrašas yra dinamiškai atnaujinamas algoritmo vykdymo eigoje (kai tik esamas sprendinys pakeičiamas nauju).

Kartu su tabu sąrašu 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, į antrinę atmintį kiekvienos iteracijos metu įtraukiami sprendiniai, likę „antri“ po aplinkos Θ_2 patikrinimo. Jeigu tabu paieškos procedūros vykdymo eigoje geriausias rastas sprendinys nesikeičia daugiau kaip $\lfloor \beta\tau \rfloor$ 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 $1 \leq \lfloor \beta\tau \rfloor \leq \tau$; realizuotame algoritme $\beta = 0{,}2$). 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, nes sumažinamas neigiamas paieškos stagnacijos poveikis. Antrinė atmintis išvaloma, pasibaigus TP procedūrai.

TP procesas yra baigiamas, kai tik atliekamas iš anksto užduotas TP vykdymo iteracijų skaičius (τ). TP algoritmo formalus pseudokodo aprašas, kuris vėlgi yra panašus į pseudokodą, pateiktą (Misevičius ir kt., 2021) publikacijoje, parodytas 16 pav.

```
procedure Tabu_paieška;
//duomenys: p – esamas sprendinys-perstatymas (kartu su atitinkamais tikslo funkcijos
//                reikšmių pokyčiais (Δ)),
//          n – uždavinio dydis (perstatymo dydis)
//rezultatai: p• – 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
01  begin
02    p• := p;  k := 1;  k' := 1; archyvo_skaitiklis := 0;
03    L := ⌊βτ⌋;   //L – tuščios eigos limitas
04    while (k ≤ τ) begin  //pagrindinis tabu paieškos iteracijų vykdymo ciklas
05      delta'min := ∞;  delta''min := ∞;  u' := 1;  v' := 2;
06       //nagrinėjama sprendinio p aplinka Θ2(p) ir ieškomas geriausias sprendinys,
07       //kuris nėra uždraustas (tabu)
08       for i := 1 to n − 1 do
09         for j := i + 1 to n do begin
10           delta := Δ(p^{i,j}, p);
11           tabu := iif((t_{i,j} ≥ k) and random() ≥ α), TRUE, FALSE);
12           aspiracija := iif(z(p) + delta < z(p•), TRUE, FALSE);
13           if ((delta < delta'min) and (tabu = FALSE)) or
14               (aspiracija = TRUE) then begin
15               if delta < delta'min then begin
16                   delta''min := delta'min;  u'' := u'; v'' := v';
17                   delta'min := delta;  u' := i; v' := j
18               endif
19               else if delta < delta''min then begin
20                         delta''min := delta;  u'' := i; v'' := j
21                    endif
22           endif
23         endfor;
24       if delta''min < ∞ then begin  //„antrojo“ sprendinio įtraukimas į
25                                     //archyvą (antrinę atmintį) Γ
26          archyvo_skaitiklis := archyvo_skaitiklis + 1;
27          Γ [archyvo_skaitiklis] ← p, Δ, u'', v''
28       endif;
29       if delta'min < ∞ then begin
30          p := p^{u',v'};  //esamas sprendinys pakeičiamas nauju sprendiniu
31          perskaičiuoti tikslo funkcijos pokyčius Δ(p^{i,j}, p),
32          i, j = 1,…, n;
33          if z(p) < z(p•) then begin  //išsaugomas geriausias
34          rastas sprendinys
35              p• := p;  k' := k  endif;
36          t_{u',v'} := k + h  //elementų p(u'), p(v') sukeitimas uždraudžiamas būsimoms
37                              //h iteracijų
38       endif;
39       if (k − k' > L) and (k < τ − L) then begin
40          //tabu paieškos paleidimas, panaudojant sprendinį iš archyvo
```

[tęsinys kitame puslapyje]

```
41          išvalyti tabu sąrašą T;
42          atsitiktinio_išrinkimo_indeksas :=
43             random(archyvo_skaitiklis * 0.8, archyvo_skaitiklis);
44          p, Δ, u'', v'' ← Γ [atsitiktinio_išrinkimo_indeksas];
45          p := p^(u'',v'');
46          perskaičiuoti tikslo funkcijos pokyčius Δ(p^(i,j), p),
47          i, j = 1,...,n;
48          t_(u'',v'') := k + h;  k' := k
49       endif;
50       k := k + 1
51     endwhile;
52     išvalyti tabu sąrašą T
53  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 $[x_1, x_2]$.

16 pav. Tabu paieškos algoritmo aprašymas

3.5.2 Iteratyviosios 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 (Misevicius, 2012). 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].

ITP algoritme yra kreipiamasi į žemesniojo lygmens algoritmą, t. y. savarankišką (angl. *self-contained*) TP algoritmą. Galima įsivaizduoti, kad šiuo atveju TP procedūra yra savotiškas iteracinio algoritmo „branduolys“. Būtent ši ir tik ši procedūra vykdo betarpišką sprendinio pagerinimą (atlieka paieškos intensifikavimą).

ITP algoritmą sudaro trys esminiai komponentai: 1) žemesnio lygmens algoritmo (TP algoritmo) panaudojimas (iškvietimas); 2) sprendinio-kandidato parinkimas mutavimui; 3) atrinkto sprendinio pertvarkymas (mutavimas).

Atkreiptinas dėmesys į tai, kad sprendinio mutavimas, t. y. diversifikavimas, yra labai svarbus komponentas ITP algoritme ir jis yra reikalingas – siekiant, kad būtų galima pereiti į kitas, nenagrinėtas sprendinių aibės (paieškos erdvės) sritis ir išvengti paieškos stagnacijos. Kas dėl sprendinio-kandidato parinkimo mutavimui, tai galimos kelios strategijos. Pvz., galima visada pasirinkti paskutinį gautą (vis kitą) pagerintą sprendinį (angl. *last best found solution*). Toks parinkimas leidžia išžvalgyti galimai didesnę sprendinių erdvės dalį. Kita galima strategija yra pagrįsta tuo, kad kiekvieną

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

kartą pasirenkamas pats geriausias duotu metu rastas sprendinys (angl. *overall best found solution*). Šis parinkimas įgalina sukoncentruoti paiešką atskirose sprendinių dalyse.

ITP algoritmo formalus aprašymas, primenantis (Misevičius ir kt., 2021) straipsnyje duotą aprašą, pateiktas 17 pav.

```
procedure Iteratyvioji_tabu_paieška;
//duomenys: p – esamas sprendinys-perstatymas
//rezultatai: p^∇ – geriausas rastas sprendinys-perstatymas (kartu su tikslo funkcijos reikšmių
//                pokyčiais (Δ))
//valdymo parametras: Q – iteratyviosios tabu paieškos vykdymo iteracijų skaičius
01  begin
02    p^∇ := p;
03    for q := 1 to Q do begin
04      vykdyti procedūrą Tabu_paieška sprendiniui p, gauti
05      (pagerintą) sprendinį p•;
06      if z(p•) < z(p^∇) then p^∇ := p•  //įsimenamas geriausias rastas sprendinys
07                                        //(kartu su TF pokyčiais)
08      if q < Q then begin
09          p := Sprendinio_kandidato_parinkimas_mutavimui(p•, p^∇);
10          vykdyti mutavimo procedūrą sprendiniui p,
11          grąžinti mutuotą sprendinį p~;
12          p := p~
13      endif
14    endfor
15  end.
```

17 pav. Iteratyviosios tabu paieškos algoritmo aprašymas

3.5.3 Hierarchinės iteratyviosios tabu paieškos algoritmas

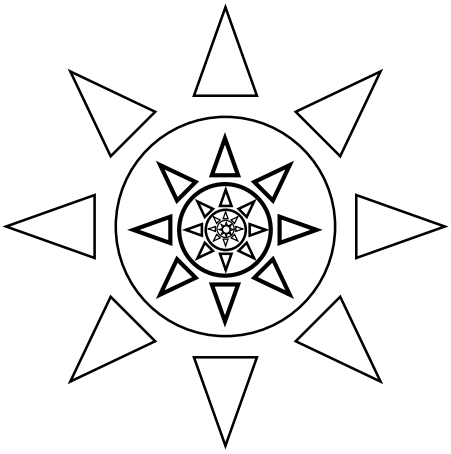
Iteratyviajai tabu paieškai, savo ruožtu, taip pat gali būti taikomas daugkartinis panaudojimas; taip gali būti gaunamas 2 (Hussin, Stützle, 2009) ir daugiau lygmenų iteratyviosios tabu paieškos algoritmas – hierarchinis iteratyviosios tabu paieškos (HITP) (angl. *hierarchical iterated tabu search*) algoritmas. Tokio tipo algoritmo skiriamoji savybė yra ta, jog jis pasižymi savo struktūros hierarchiškumu, tiksliau, panašumu į save (angl. *self-similarity*). Panašumas į save, kaip žinoma, yra turbūt vienas iš fundamentaliausių gamtos principų. Taigi, hierarchinį ITP algoritmą (k lygmenų ITP algoritmą) sudaro keli panašūs į save struktūriniai lygmenys (k savi-panašių struktūrinių lygmenų). Aukštesniojo (k-ojo) lygmens algoritmas kreipiasi į žemesniojo (k–1-ojo) lygmens algoritmą, kuris kreipiasi į k–2 lygmens algoritmą ir t. t. iki pirmojo lygmens (žr. 18 pav.). Pirmojo lygmens algoritmas jau betarpiškai naudoja TP algoritmą, t. y. savarankišką TP algoritmą (nulinio lygmens algoritmą). Būtent ši ir tik ši procedūra vykdo betarpišką sprendinio pagerinimą (atlieka paieškos intensifikavimą).

Kaip ir iteracinės tabu paieškos atveju, algoritmo lygmenys yra tos pačios struktūros (esamo lygio algoritmą sudaro trys ingredientai: sprendinio lokalusis

pagerinimas (optimizavimas), panaudojant savi-panašų, vienu lygmeniu žemesnį hierarchinį algoritmą; sprendinio-kandidato parinkimas mutavimui; atrinkto sprendinio pertvarkymas (mutavimas)). Nors atskirų lygmenų struktūra yra išties paprasta, tačiau šių lygmenų tinkamas sujungimas leidžia žymiai padidinti paieškos našumą. Aišku, paieškos laikas išauga, bet tai yra kompensuojama aukštesne rezultatų kokybe. Matyti, jog sprendinio mutavimas k lygmenų algoritme vykdomas kiekviename lygmenyje – skirtingai nuo TP procedūros („branduolio“ procedūros), kuri atliekama tik žemiausiame lygmenyje. Taigi, hierarchinio iteracinio algoritmo atveju santykinai dominuoja paieškos diversifikavimo faktorius, o tai ir yra labai svarbu, vykdant paiešką didžiulėje sprendinių erdvėje su daug lokaliai optimalių sprendinių. Sprendinio-kandidato parinkimui galioja tos pačios strategijos, kaip ir ITP paieškai. Be to, gali būti naudojamos ir kombinuotos strategijos.

Daugelio lygių HITP algoritmo aprašymas pateiktas 19 pav.

Genetiniame hierarchiniame algoritme buvo realizuoti 7 ($k = 7$) iteratyviosios tabu paieškos lygmenys. Šis konkretus lygmenų skaičius parinktas, atsižvelgiant į išankstinius, preliminarius eksperimentus, taip pat tam tikrus intuityvius bendrus samprotavimus, jog šis skaičius neturėtų būti nei per mažas, nei per didelis.



18 pav. Savi-panašių lygmenų struktūra

```
procedure k-lygių_hierarchinė_iteratyvioji_tabu_paieška;
//duomenys: p°– pradinis sprendinys-perstatymas
//rezultatai: p◊ – geriausas rastas sprendinys-perstatymas
//valdymo parametras: Q_k – k-ojo lygio tabu paieškos vykdymo iteracijų skaičius
01  begin
02    apskaičiuoti tikslo funkcijos pokyčius Δ((p°)^{i,j}, p°),
03    i, j = 1,...,n;
04    inicializuoti tabu sąrašą T;
05    p := p°;  p◊ := p°;
06    for q_k := 1 to Q_k do begin
07      vykdyti procedūrą
08      k–1-lygio_hierarchinė_iteratyvioji_tabu_paieška
09      sprendiniui p, gauti (pagerintą) sprendinį p▽;
10      if z(p▽) < z(p◊) then p◊ := p▽;  //įsimenamas geriausias
11      rastas sprendinys
12      if q_k < Q_k then begin
13         p := Sprendinio_kandidato_parinkimas_mutavimui(p▽, p◊);
14         vykdyti mutavimo procedūrą sprendiniui p,grąžinti
15         mutuotą sprendinį p~;
16         p := p~
17      endif
18    endfor
19  end.
```

Pastaba. Kai k yra lygus 1, tai HITP algoritmo aprašymas tampa panašus į ITP algoritmo aprašymą, kur kreipimasis į savarankišką TP algoritmą laikomas tiesiog kreipimusi į nulinio lygio algoritmą.

19 pav. k lygių hierarchinės iteratyviosios tabu paieškos algoritmo aprašymas

3.5.4 Tabu paieškos mutavimo procedūros

Realizuoto genetinio hierarchinio algoritmo ypatumas yra tas, kad sprendinių mutavimas yra atliekamas hierarchininės iteratyviosios tabu paieškos algoritmo viduje, bet ne tiesiogiai genetiniame algoritme, kaip tai yra įprasta daugeliu atvejų. Tačiau, kaip rodo eksperimentų rezultatai, tai nesumažina genetinio algoritmo efektyvumo, bet kaip tik atvirkščiai.

Mutavimo procesas (Lim ir kt., 2017) sprendinių-perstatymų atveju gali būti apibrėžiamas, panaudojant formalų operatorių ϕ: $\Pi_n \to \Pi_n$. Taigi, jeigu $p^{\sim} = \phi(p)$, tai $p^{\sim} \in \Pi_n$; be to, $p^{\sim} \neq p$. Galima naudoti ir labiau formalizuotą operatorių $\phi(p, \xi)$: $\Pi_n \times \mathrm{N} \to \Pi_n$, kuris esamą sprendinį p transformuoja į naują sprendinį $p^{\sim}$ ir tokį, jog $\delta(p, p^{\sim}) = \delta(p, \phi(p)) = \xi$. 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:

$$p^{\sim} = \varphi(\phi(\phi(\phi(\phi(\phi(p, u_1, u_2), u_2, u_3) \ldots), u_{t-1}, u_t), u_t, u_{t+1}) \ldots) = \\ = (((((p^{u_1,u_2})^{u_2,u_3})^{\cdots})^{u_{t-1},u_t})^{u_{t,t+1}})^{\cdots}. \quad (25)$$

Dydis (parametras) ξ vadinamas mutavimo laipsniu (stiprumu). Akivaizdu, jog $2 \leq \xi \leq n$. Taigi, po mutavimo pakinta $\frac{100\xi}{n}$ procentų sprendinio elementų. Parametras ξ yra iš esmės vienintelis kiekybinis mutavimo procesą reguliuojantis faktorius. Kuo

didesnė ξ reikšmė, tuo daugiau sprendinio elementų sukeičiama (tuo didesniu mastu pakinta sprendinys, tuo labiau mutuotas sprendinys „nutolsta“ nuo esamo sprendinio), ir atvirkščiai. Galima laikyti, kad mutavimo operatorius $\phi(p, \xi)$ transformuoja esamą sprendinį p į kitą sprendinį, priklausantį sprendinio p aplinkai $\Theta_{\xi}(p)$. Dar sakoma, jog atliekamas ξ-perėjimas iš sprendinio p į sprendinį $p^{\sim}$.

Mutavimo procesui pagal jo prigimtį yra būdingas grynai atsitiktinis pobūdis, mutavimas yra indeterminuotas procesas; tuo mutavimas skiriasi nuo kryžminimo. Mutavimo proceso metu generuojamos, bendrai kalbant, naujos sprendinių savybės, nauja informacija; tuo tarpu kryžminimo procedūros paprastai tik (re)kombinuoja informaciją, esančią pirmtakuose. Skirtingai negu kryžminimo procedūrose, kur yra paveldimas sukauptas genetinis kodas iš dviejų pirmtakų ir daugiau niekur kitur, mutavimo procedūrose pagrindinis dalykas yra ne informacijos perdavimas, o naujos informacijos, kurios prieš tai dar nebuvo, įnešimas, sukūrimas. Šiuo požiūriu mutavimo procesas yra labiau naujumo generavimo procesas, labiau kuriantysis procesas, jeigu lygintume su kryžminimu.

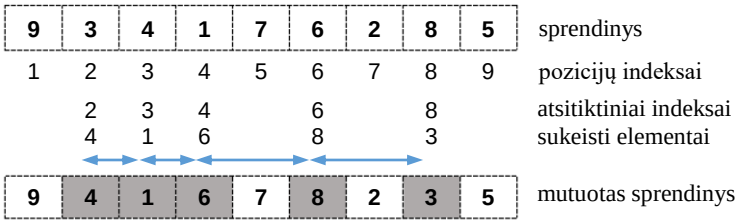
Mutavimo proceso algoritminis realizavimas nėra labai sudėtingas. Teorinis pagrindas yra mutavimo operatorius ϕ, aprašytas anksčiau. Atsižvelgiant į mutavimo algoritminio realizavimo skirtumus (ypatumus), galima sudaryti įvairių tipų mutavimo procedūras.

Toliau trumpai aprašomos sudarytos ir eksperimentuose naudotos mutavimo procedūros.

3.5.4.1 Atsitiktiniai poriniai elementų sukeitimai

Ši mutavimo procedūra yra gana nesudėtinga (Merz, Freisleben, 1997) (žr. 20 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 išvengti pasikartojančių atsitiktinių skaičių generavimo, yra iš anksto sugeneruojamas perstatymas $s = (s_{(1)}, s_{(2)}, \dots, s_{(n)})$, sudarytas iš atsitiktinai sumaišytų sveikų skaičių nuo 1 iki n. Skaičių poros $(s_{(1)}, s_{(2)})$ tokiu būdu nurodo perstatymo $p^{\sim}$ atsitiktinių pozicijų indeksus.) 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 $\xi - 1$ skaičių kartų; čia ξ – algoritmo tyrėjo / vartotojo iš anksto nustatyta (norima) mutavimo laipsnio reikšmė. Mutavimo procedūros rezultatas – mutuotas sprendinys $p^{\sim}$, tenkinantis sąlygą $\delta(p, p^{\sim}) = \xi$. Aišku, kad formuojant mutuotą sprendinį-perstatymą yra užtikrinamas reikalavimas, jog sprendinyje-perstatyme jokie elementai nesikartoja, t. y. $p^{\sim} \in \Pi_n$. Mutavimo proceso stiprumas kontroliuojamas parametro ξ pagalba ($2 \leq \xi \leq n$).

Šios mutavimo procedūros principo pagrindu galima sudaryti ir daug įvairių kitų modifikuotų mutavimo procedūrų.



20 pav. Mutavimo procedūros pavyzdžio iliustracija ($\boldsymbol{n = 9}$, $\xi = \mathbf{5}$)

3.5.4.2 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 (angl. *random key*) (Bean, 1994). Atsitiktinis raktas yra papildomas atsitiktinių indeksų nuo 1 iki ξ masyvas (rinkinys $r = (r_{(1)}, r_{(2)}, \dots, r_{(\xi)})$). Atsitiktinio rakto reikšmės nurodo, kokie anksčiau sukeisti elementai su kokiais elementais maišomi. Taip bandoma gauti labiau „gilesnį“ mutuotą sprendinį, daugiau randomizuotą elementų išmaišymą. Tuomet TP proceso eigoje ne taip paprasta grįžti į ankstesnę situaciją, tik atliekant porinius elementų sukeitimus.

3.5.4.3 Mutavimas panaudojant priešines (opozicines) reikšmes

Šioje mutavimo procedūroje sprendinio p klone $p^{\sim}$ atsitiktinai parenkama pozicija, tarkime, k. Sprendinio $p^{\sim}$ esama elemento reikšmė $e = p^{\sim}(k)$ pakeičiama tokia priešine (opozicine) (angl. *opposition*) reikšme: $o = \Big(\big((p^{\sim}(k) + n/2 - 1) \bmod n\big) - 1\Big) + 1$, 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.

3.5.4.4 Maksimalaus elementų atstumo mutavimas

Duotoje procedūroje sukeičiamų sprendinio elementų pozicijų indeksai generuojami taip, jog atstumas (λ) tarp tų pozicijų būtų kaip įmanoma didesnis. Panaudota tokia sukeičiamų sprendinio elementų indeksų reikšmių k_l generavimo formulė:

$$k_l = \big\lfloor \big((\lambda q_l + \xi - 1) \bmod n\big) + 1 \big\rfloor, \tag{26}$$

čia $\lambda = n/\xi$, ξ – (pseudo-)atsitiktinis realusis skaičius iš intervalo $[0, 1]$, $q_l = (q_{l-1} \bmod n) + 1$, $l = 1, 2, \dots, \xi$; pradinė q_l reikšmė (q_0) – tai atsitiktinis sveikas skaičius iš intervalo $[1, n]$. Tokiu būdu yra sugeneruojama ξ atsitiktinių indeksų reikšmių (čia ξ yra mutavimo laipsnis).

3.5.4.5 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. Sugeneruoti skaičiai (kartu su juos atitinkančiais indeksais (angl. *smallest position values*)) išrikiuojami didėjimo tvarka (Tasgetiren ir kt., 2009). Išrikiuotus atsitiktinius skaičius atitinkantys išmaišyti indeksai ir nurodo, kurias sprendinio p elementų poras reikia sukeisti.

3.5.4.6 Modifikuotas atsitiktinių elementų sukeitimų mutavimas – II

Procedūros tikslas – nuosekliai, vienas po kito generuojant atsitiktinius sveikus skaičius iš intervalo $[0, 1]$, iš karto gauti sprendinio p sukeičiamų elementų porų indeksus. Tačiau, kaip pastebėta, atsitiktiniai sveikieji skaičiai gali dubliuotis. Norint išvengti dubliavimosi, sugeneruoti atsitiktiniai sveikieji skaičiai surikiuojami. Surikiuotus skaičius atitinkantys indeksai nurodo reikalingus sumaišyti elementus.

3.5.4.7 Kombinuoto mutavimo procedūra

Kombinuoto mutavimo procedūroje yra nuosekliai sujungtos dvi mutavimo procedūros. Iš pradžių nustatomi atsitiktiniai perstatymo elementų indeksai. Po to nustatyti elementai yra mutuojami, panaudojant priešines reikšmes.

3.5.4.8 Godžios adaptyvios paieškos mutavimas

Šios mutavimo procedūros principas yra kitoks negu prieš tai nagrinėtų. Ypatumas yra tas, jog iš pradžių sprendinys „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 pilną 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). Pirmos fazės metu yra anuliuojama ξ elementų, t. y. tiek, koks yra mutavimo laipsnis. Tuo tarpu, antroje fazėje taikomas godusis konstravimo algoritmas. Šis algoritmas iš visų galimų $\xi!$ variantų atrenka tą, kuriam apskaičiuotas tikslo funkcijos įvertinimas (tam tikra tikslo funkcijos ribos reikšmė) yra minimalus. Reikšmė ξ yra apribota ($\xi \leq 7$), kad labai neišaugtų mutavimo procedūros vykdymo laikas.

3.5.4.9 Godžios randomizuotos adaptyvios paieškos mutavimas

Ši mutavimo procedūra primena prieš tai aprašytą. Skirtumas tas, jog dalinio sprendinio rekonstravimo fazėje yra naudojamas godžios randomizuotos adaptyvios paieškos (GRASP) algoritmas (Li ir kt., 1994), norint gauti pagerintą sprendinį.

3.5.4.10 Randomizuotos lokalios paieškos mutavimas

Šiuo atveju iš pradžių vykdomi atsitiktiniai poriniai elementų sukeitimai. Po to atliekama randomizuota lokalioji paieška (Hoos, Stützle, 2004), 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 likusios mutavimo procedūros jau nėra tokios universalios (nepriklausančios nuo sprendžiamo uždavinio), kaip anksčiau aprašytos, nes turi būti įvertinama ir tikslo funkcija bei kitos specifinės konkretaus uždavinio savybės.

3.5.4.11 Randomizuotos dalinės lokalios paieškos mutavimas

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

3.5.4.12 Išsklaidytas (išsklaidytos tabu paieškos) mutavimas

Šiuo atveju mutavimo procedūros vaidmenyje yra tiesiog pertvarkytas tabu paieškos algoritmas, kuriame atliekami tik perėjimai tarp „antrųjų“ sprendinių. Atliekamas ribotas TP vykdymo iteracijų skaičius (ξ iteracijų). Taigi, mutavimas tampa ne tiek atsitiktinis, kiek godus (kvazi-determinuotas).

Atliekant mutavimo procedūras, pasikeičia sprendinio TF reikšmė ir TF reikšmių pokyčiai, ir yra labai svarbu šiuos pokyčius greitai perskaičiuoti. Kaip žinoma, TF pokyčiai, sukeitus du elementus, apskaičiuojami panaudojant $O(n^2)$ operacijų. Kadangi visos minėtos mutavimo procedūros atlieka apytikriai ξ ar panašų (proporcingą) skaičių elementų sukeitimų, tai visų mutavimo procedūrų sudėtingumas yra proporcingas $O(\xi n^2)$.

Mutavimo laipsnis ξ atlieka svarbų vaidmenį mutavimo procedūroje, tuo pačiu ir ją naudojančiame iteraciniame tabu paieškos algoritme. Turi būti pasirenkamas tam tikras kompromisinis variantas tarp dviejų ekstremalių situacijų: 1) reikšmė ξ yra (labai) maža; 2) reikšmė ξ yra (labai) didelė (pvz., artima n). Pirmuoju atveju, mutavimas negarantuotų pakankamai „nutolusio“ (nuo duotojo) sprendinio generavimo, o tai sąlygotų 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 eigoje lokaliai optimaliuose sprendiniuose sukauptą naudingą informaciją.

3.6 Pradinės populiacijos konstravimas

Pradinė populiacija konstruojama, atsižvelgiant į populiacijos pradinio pagerinimo varianto parametrą $PPGV$. Šis parametras gali įgyti tris reikšmes: 1, 2 ir 3. Priklausomai nuo šios reikšmės, pradinės populiacijos konstravimui naudojamas hierarchinis iteratyviosios tabu paieškos algoritmas ($PPGV = 1$) arba pavaldusis genetinis algoritmas ($PPGV = 2$), arba dviejų lygių pavaldusis genetinis algoritmas ($PPGV = 3$). Nors antrasis ir trečiasis variantai yra labiau imlūs skaičiavimų laiko atžvilgiu, tačiau tai kompensuojama aukštesnės kokybės pradinės populiacijos sudarymu.

3.7 Sprendinių-tėvų atrinkimas

Sprendinių-tėvų atrinkimas atliekamas, pasinaudojant sprendinių rango (kokybės) įvertinimo taisykle. Sakykime, kad visi populiaciją sudarantys sprendiniai yra surūšiuoti tikslo funkcijos reikšmių didėjimo tvarka, pradedant geriausiu sprendiniu pirmoje pozicijoje ir baigiant blogiausiu sprendiniu paskutinėje pozicijoje (pozicijoje PD). Tėvų parinkimui taikoma tokia taisyklė (Tate, Smith, 1995). Eilinio parenkamo „tėvo“ pozicija u sutvarkytoje populiacijoje nustatoma pagal formulę:

$$u = \lfloor v^{\sigma} \rfloor; \tag{27}$$

čia v yra atsitiktinis sveikas skaičius iš intervalo $[1, PD^{1/\sigma}]$, PD – populiacijos dydis, o σ – tam tikras, realus skaičius iš intervalo $[1, 2]$ (σ galima traktuoti kaip sprendinių-tėvų atrinkimo koeficientą). Akivaizdu, jog kuo geresnis individas, tuo didesnė tikimybė, kad jis bus atrinktas tolesniam kryžminimui.

Kai yra naudojamas pirmas sprendinių atrinkimo variantas ($TAV = 1$), tuomet sprendinių atrinkimo koeficientas σ yra lygus 1, ir gaunamas tiesiog atsitiktinis tėvų atrinkimas. Antruoju atrinkimo atveju ($TAV = 2$) galioja nelygybė $1 < \sigma \leq 2$.

3.8 Populiacijos atnaujinimas

Populiacija atnaujinama, atsižvelgiant į populiacijos atnaujinimo variantą PAV. Dvi pagrindinės populiarios populiacijos atnaujinimo strategijos literatūroje yra žinomos kaip „$\mu + \lambda$“ atnaujinimas ir „μ, λ“ atnaujinimas (Sivanandam, Deepa, 2008). Tarkime, kad populiacijos dydis yra lygus μ, o naujų sukurtų palikuonių skaičius yra lygus λ (GH algoritme $\mu = PD$, $\lambda = 1$). Tuomet pirmuoju atveju atrenkama (paliekama) μ geriausių sprendinių iš aibių sąjungos $P_{\mu} \cup P_{\lambda}$, čia P_{μ} žymi esamos generacijos sprendinių aibę (populiaciją), o P_{λ} yra naujų sukurtų palikuonių aibė. Jeigu $\lambda = 1$, tai sukuriamas tik vienas palikuonis, kuris pakeičia esamos populiacijos blogiausią individą (jeigu sukurtas palikuonis yra blogesnis už blogiausią populiacijos individą, tuomet populiacija lieka nepakeista). Antruoju atveju naujas sukuriamas palikuonis pakeičia palikuonio vieną iš tėvų, neatsižvelgiant į palikuonio vertę.

GH algoritme realizuota modifikuota „$\mu + \lambda$“ atnaujinimo taisyklė, kuri formaliai žymima kaip ir „$\mu + \lambda, \varepsilon$“. Šiuo atveju, prieš įtraukiant naują palikuonį į populiaciją, atsižvelgiama į minimalaus atstumo tarp sprendinių kriterijų. Atstumas tarp sprendinių apskaičiuojamas pagal (15) formulę ($\delta(p_1, p_2) = |\{i: p_1(i) \neq p_2(i)\}|$). Tuomet jeigu minimalus atstumas tarp palikuonio ir esamų populiacijos sprendinių yra didesnis už apibrėžtą atstumo slenkstį ε, tai palikuonis ignoruojamas (neįtraukiamas į populiaciją) – nebent jis būtų geresnis už geriausią populiacijos individą. (GH algoritme naudota $\varepsilon = 0{,}25n$.) Tai ir yra pirmasis populiacijos atnaujinimo variantas ($PAV = 1$). Antras realizuotas atnaujinimo variantas ($PAV = 2$) skiriasi nuo pirmojo tuo, kad palikuonis – jeigu jis yra geresnis už geriausią populiacijos individą – pakeičia tą geriausią individą, bet ne blogiausią individą, kaip yra pirmojo varianto atveju. Trečiojo populiacijos atnaujinimo varianto atveju ($PAV = 3$) palikuonis pakeičia arba blogiausią populiacijos individą, arba vieną iš

individų, esančių „elitinėje“ populiacijos dalyje, t. y. populiacijos dalyje, kurioje yra pusė geriausių populiacijos individų.

3.9 Populiacijos perkrovimas

Populiacijos perkrovimas yra gana svarbus genetinio algoritmo komponentas, kadangi jis įgalina paieškos (tuo pačiu ir populiacijos) diversifikavimą algoritmo stagnavimo atvejais (Misevicius, 2008). Perkrovimo principo esmė ta, jog esama „stagnacinė“ populiacija yra savotiškai „renovuojama“ (regeneruojama). GH algoritme perkrovimas vykdomas, kai populiacijos sprendiniai visiškai nesikeičia L nuoseklių generacijų ($L = \omega N_{gen}$, ω – populiacijos stagnacijos koeficientas, N_{gen} – bendras generacijų skaičius; standartinė ω reikšmė yra $0{,}07$). Išbandyta keletas populiacijos perkrovimo variantų. Paprasčiausias variantas yra kai pats perkrovimas neatliekamas, o tik padidinamas mutavimo stiprumo laipsnis tabu paieškos algoritme ($PPV = 1$). Kitas variantas yra toks, kai tiesiog iš naujo sugeneruojama nauja populiacija, kurios visi sprendiniai pagerinami, panaudojant HITP algoritmą ($PPV = 2$). Daug žadantis yra populiacijos perkrovimo variantas, kuomet populiacija ne generuojama iš naujo, bet atliekamas visų esamos populiacijos sprendinių mutavimas (multi-mutavimas). Mutavimui galima panaudoti bet kurią iš anksčiau aprašytų procedūrų (žr. 3.5.4 sk.). GH algoritme buvo išbandyta modifikuoto atsitiktinių elementų sukeitimų mutavimo procedūra (žr. 3.5.4.5 sk.) ($PPV = 3$), taip pat išsklaidyto mutavimo procedūra (žr. 3.5.4.12 sk.) ($PPV = 4$) bei kombinuoto mutavimo procedūra ($PPV = 5$), kurioje apjungtos modifikuoto atsitiktinių elementų sukeitimų mutavimo procedūra ir išsklaidyto mutavimo procedūra. Taip pat buvo eksperimentuota su papildoma multi-mutavimo procedūra ($PPV = 6$), kurioje mutuojami ne tik esamos populiacijos sprendiniai, bet ir tabu paieškos algoritmo eigoje gaunami sprendiniai („antrieji“ sprendiniai). Mutuojami tik atrinkti geriausi antrieji sprendiniai. Visose multi-mutavimo procedūrose yra atsižvelgiama į minimalaus atstumo kriterijų (esant nepatenkintam minimalaus atstumo kriterijui, į populiaciją įtraukiamas atsitiktiniu būdu sugeneruotas sprendinys). Be šių populiacijos perkrovimo variantų, buvo išbandyti dar du populiacijos perkrovimo būdai, kuomet perkrovimui panaudojamas pavaldusis genetinis algoritmas ($PPV = 7$ ir $PPV = 8$).

3.10 Apibendrinamosios pastabos

Šiame skyriuje iš pradžių pateiktas kvadratinio paskirstymo uždavinio formalus aprašymas ir praktiniai taikymai. Šio uždavinio sprendimui pristatomas hibridinis genetinis hierarchinis algoritmas ir jo variantai. Esminės naujosios genetinio hierarchinio algoritmo savybės yra: originali pačio genetinio algoritmo struktūra (architektūra), integruotas į GA sudėtį hierarchinės iteratyviosios tabu paieškos algoritmas, naujoviška universaliojo kryžminimo procedūra.

Skyriuje aprašyta 12 kryžminimo procedūrų, kurios visos buvo programiškai realizuotos, po to eksperimentiškai ištirtos, siekiant nustatyti tinkamiausią jų. Eksperimentų metu nustatyta, jog efektyviausios yra universaliojo, vieno taško ir suliejimo kryžminimo procedūros.

Šiame skyriuje taip pat pristatomas hierarchinės iteratyviosios tabu paieškos algoritmas, skirtas sprendinių pagerinimui GA eigoje. HITP algoritmą, savo ruožtu, sudaro iteratyviosios tabu paieškos procedūra ir pats savarankiškas tabu paieškos algoritmas, kuris betarpiškai realizuoja sprendinių pagerinimą.

Iteratyviosios tabu paieškos efektyvumui padidinti yra naudojamos sprendinių mutavimo procedūros, kurių – panašiai kaip ir kryžminimo procedūrų – buvo nagrinėta vėlgi 12. Mutavimo procedūros yra įkomponuotos betarpiškai į ITP algoritmą. Eksperimentais nustatyta, jog efektyviausios yra išsklaidytos paieškos ir randomizuotos dalinės paieškos mutavimo procedūros.

Skyriaus pabaigoje taip pat yra aprašyti kiti svarbūs GA komponentai: pradinės populiacijos konstravimas, sprendinių-tėvų atrinkimas, populiacijos atnaujinimas bei populiacijos perkrovimas.

4 EKSPERIMENTINIAI TYRIMAI

Buvo atlikti nuodugnūs kompiuteriniai-eksperimentiniai tyrimai, kurių tikslas nustatyti tinkamiausius ir perspektyviausius sudaryto genetinio hierarchinio algoritmo variantus.

Toliau atskirus GH algoritmo modifikacijas, t. y. algoritmo egzempliorius, apibrėžiančius modifikuojamus variantus (vieną ar daugiau) dar vadinsime (algoritmo) konfigūracijomis.

Kompiuteriniuose-eksperimentiniuose tyrimuose panaudoti kvadratinio paskirstymo uždavinio testinių duomenų pavyzdžiai iš viešosios elektroninės KP uždavinio testinių duomenų bibliotekos QAPLIB (https://coral.ise.lehigh.edu/data-sets/qaplib/; Burkard ir kt., 1997), taip pat publikacijų (De Carvalho, Rahmann, 2006) bei (Drezner ir kt., 2005; žr. taip pat http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap.html). Eksperimentai atlikti iš viso su 114 testinių duomenų pavyzdžių. Eksperimentuose naudoti šie skirtingi testinių duomenų tipai: 1) atsitiktiniai, nestruktūrizuoti duomenys (`rou020`, `tai010a`, `tai012a`, `tai015a`, `tai017a`, `tai020a`, `tai025a`, `tai030a`, `tai035a`, `tai040a`, `tai050a`, `tai060a`, `tai080a`, `tai100a`); 2) atsitiktiniai duomenys su reguliariais atstumais (`had020`, `nug030`, `scr020`, `sko042`, `sko049`, `sko056`, `sko064`, `sko072`, `sko081`, `sko090`, `sko100a..sko100f`, `tho030`, `tho040`, `tho150`, `wil050`, `wil100`); 3) realaus pasaulio, struktūrizuoti duomenys (`chr025a`, `els019`, `esc032a..esc032h`, `esc064a`, `esc128`, `kra030a`, `kra030b`, `ste036a..ste036c`, `tai064c`); 4) pseudo-atsitiktiniai duomenys, imituojantys realaus pasaulio duomenis (`tai010b`, `tai012b`, `tai015b`, `tai020b`, `tai025b`, `tai030b`, `tai035b`, `tai040b`, `tai050b`, `tai060b`, `tai080b`, `tai100b`, `tai150b`); 5) duomenys su žinomais optimaliais sprendiniais (`lipa020a`, `lipa020b`, `lipa030a`, `lipa030b`, `lipa040a`, `lipa040b`, `lipa050a`, `lipa050b`, `lipa060a`, `lipa060b`, `lipa070a`, `lipa070b`, `lipa080a`, `lipa080b`, `lipa090a`, `lipa090b`); 6) sudėtingi (specialūs) duomenys iš (De Carvalho, Rahmann, 2006) straipsnio (`bl036`, `bl049`, `bl064`, `bl081`, `bl100`, `bl121`, `bl144`, `ci036`, `ci049`, `ci064`, `ci081`, `ci100`, `ci121`, `ci144`); 7) sudėtingi (specialūs) duomenys iš (Drezner ir kt., 2005) straipsnio (`dre015`, `dre018`, `dre021`, `dre024`, `dre028`, `dre030`, `dre042`, `dre056`, `dre072`, `dre090`, `dre110`, `dre132`, `tai027e1`, `tai027e2`, `tai027e3`, `tai045e1`, `tai045e2`, `tai045e3`, `tai075e1`, `tai075e2`, `tai075e3`).

Eksperimentuose naudotas 3,2 GHz taktinio dažnio stacionarus asmeninis kompiuteris. (Kompiuterio charakteristikos: operacinė sistema – Windows 10 (64 bitų), CPU modelis – Intel(R) Core(TM) i7-8650U, branduolių skaičius – 4, operatyvioji atmintis (RAM) – 16 GB.) Buvo naudota sudaryto algoritmo programinė realizacija C# programavimo kalba.

Atliekant eksperimentus, GH algoritmo 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 ($\bar{\theta}$) apskaičiuojamas pagal formulę:

$$\bar{\theta} = 100(\bar{z} - z_{G\check{Z}R})/z_{G\check{Z}R}[\%], \quad (28)$$

čia $\bar{z}$ yra vidutinė tikslo funkcijos reikšmė, gauta atlikus K pakartotinių algoritmo vykdymų, $z_{G\check{Z}R}$ žymi geriausią žinomą tikslo funkcijos reikšmę, atitinkančią (pseudo)optimalų, t. y. geriausią žinomą sprendinį (GŽS) iš bibliotekos QAPLIB. Visuose eksperimentuose K reikšmė yra lygi 10.

Genetinio algoritmo bendras iteracijų, t. y. generacijų skaičius, lygus 100, populiacijos dydis, t. y. populiacijos individų (sprendinių) skaičius, lygus 10 (jeigu konkrečiame eksperimente nenurodyta kitaip).

Visuose hierarchinės iteratyviosios tabu paieškos lygiuose vykdoma po 2 iteracijas ($Q = Q_1 = Q_2 = Q_3 = Q_4 = Q_5 = Q_6 = Q_7 = 2$), išskyrus nulinį lygį, t. y. patį tabu paieškos algoritmą, kurio iteracijų skaičius (τ) yra lygus 20 (jeigu nenurodyta kitaip).

Kalbant apie mutavimo parametrą ξ, tai šio parametro reikšmė (po preliminarių eksperimentų) prilyginta $\lfloor 0{,}2n \rfloor$, čia n – uždavinio apimtis.

Genetinio algoritmo variantų testavimui panaudoti šie bibliotekoje QAPLIB esantys testavimo duomenų pavyzdžiai: `bl049`, `bl064`, `ci049`, `ci064`, `dre042`, `dre056`, `lipa070a`, `lipa070b`, `sko056`, `sko064`, `tai035a`, `tai035b`, `tai040a`, `tai040b`, `tai045e1`, `tai050a`, `tai050b`, `tai060a`, `tai060b`, `wil050`. Testinių duomenų pavyzdžiai yra parinkti taip, jog jie yra vidutinio dydžio. Testiniai duomenys yra įvairių tipų, pradedant atsitiktiniu būdu sugeneruotais pavyzdžiais ir baigiant pavyzdžiais, imituojančiais duomenis iš „realaus pasaulio“ (realių praktinių taikymų).

4.1 Eksperimentai su genetinio algoritmo kryžminimo procedūromis

Vieni iš svarbių eksperimentų yra skirti nustatyti pažangiausius GA kryžminimo procedūrų variantus. Šie tyrimai ir buvo atlikti pradiniame eksperimentavimo etape. Eksperimentuose lygintos šios kryžminimo procedūros:

1) vieno taško kryžminimo procedūra – 1TKP;
2) dviejų taškų kryžminimo procedūra – 2TKP;
3) tolygiojo pasiskirstymo kryžminimo procedūra – TPKP;
4) sumaišymo kryžminimo procedūra – SUMKP;
5) dalinio atvaizdavimo kryžminimo procedūra – DAKP;
6) sukeitimų kryžminimo procedūra – SUKKP;
7) euristinė sukeitimų kryžminimo procedūra – ESUKKP;
8) modifikuota euristinė sukeitimų kryžminimo procedūra – MESUKKP;
9) ciklinio kryžminimo procedūra – CKP;
10) suliejimo kryžminimo procedūra – SULKP;

11) universaliojo kryžminimo procedūra – UKP;

12) daugelio tėvų kryžminimo procedūra – DTKP.

Kryžminimo procedūrų palyginimo rezultatai (sprendinių vidutinių nuokrypių reikšmės) pateikti 1 lentelėje.

Esant tam pačiam testinių duomenų dydžiui n, kompiuterio centrinio procesoriaus laikas yra maždaug vienodas visoms lygintoms kryžminimo procedūroms, taigi, kryžminimo algoritmų efektyvumą iš esmės indikuoja tik TF santykinis nuokrypis. Tokiu būdu, iš 1 lentelėje pateiktų rezultatų matyti, kad perspektyviausios yra universaliojo, vieno taško bei suliejimo kryžminimo procedūros.

1 lentelė. Kryžminimo procedūrų palyginimo rezultatai

Vidutinis santykinis procentinis nuokrypis												
Kryž. var.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bl049	0,730	0,765	0,712	0,756	0,756	0,812	0,792	0,730	0,765	0,712	0,668	0,853
bl064	1,009	0,775	0,768	1,222	1,062	0,795	1,136	1,096	0,882	0,962	0,962	1,416
ci049	0,004	0,002	0,009	0,000	0,012	0,004	0,000	0,000	0,003	0,013	0,005	0,003
ci064	0,092	0,086	0,103	0,087	0,081	0,097	0,066	0,055	0,085	0,068	0,098	0,110
dre042	8,770	15,052	13,665	11,990	11,126	18,770	9,738	8,586	14,058	14,607	7,408	7,696
dre056	28,785	28,674	36,206	37,661	35,470	39,282	38,122	38,471	29,963	24,199	26,298	46,335
lipa070a	0,165	0,165	0,165	0,165	0,165	0,165	0,165	0,165	0,165	0,165	0,165	0,165
lipa070b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
sko056	0,002	0,001	0,018	0,000	0,017	0,021	0,002	0,000	0,002	0,000	0,001	0,014
sko064	0,006	0,000	0,022	0,000	0,000	0,016	0,000	0,000	0,000	0,000	0,006	0,015
tai035a	0,355	0,416	0,332	0,263	0,233	0,506	0,313	0,239	0,386	0,252	0,484	0,215
tai035b	0,000	0,000	0,000	0,000	0,000	0,019	0,000	0,000	0,000	0,000	0,000	0,000
tai040a	0,483	0,417	0,556	0,477	0,482	0,686	0,464	0,462	0,513	0,622	0,601	0,534
tai040b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai045e1	3,253	1,366	2,944	1,307	1,285	2,682	0,172	1,597	3,506	2,327	1,482	6,323
tai050a	0,810	0,834	0,872	0,742	0,712	0,960	0,784	0,884	0,829	0,797	0,912	0,838
tai050b	0,000	0,033	0,033	0,000	0,000	0,000	0,000	0,033	0,033	0,066	0,033	0,113
tai060a	0,819	0,826	0,904	0,894	0,858	0,976	0,899	0,865	0,971	0,762	0,888	0,901
tai060b	0,037	0,000	0,000	0,000	0,037	0,000	0,000	0,000	0,000	0,000	0,000	0,040
wil050	0,002	0,003	0,013	0,007	0,007	0,011	0,008	0,000	0,005	0,007	0,011	0,007
Vidurkis:	2,266	2,471	2,866	2,779	2,615	3,290	2,633	2,659	2,608	2,278	2,001	3,279

Pastaba. Visais kryžminimo atvejais taikytas pirmas mutavimo variantas (atsitiktiniai poriniai elementų sukeitimai).

4.2 Eksperimentai su mutavimo procedūromis

Tęsiant eksperimentus, GH algoritmo ir tuo pačiu mutavimo procedūrų veikimo įvertinimo kriterijus yra toks pats, kaip ir tiriant kryžminimo procedūras.

Eksperimentuose lyginti šie mutavimo procedūrų variantai:

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) godžios adaptyvios paieškos mutavimas – M8;
9) godžios randomizuotos adaptyvios paieškos mutavimas – M9;
10) randomizuotos lokalios paieškos mutavimas – M10;
11) randomizuotos dalinės lokalios paieškos mutavimas – M11;
12) išsklaidytas mutavimas – M12.

Atliktų eksperimentų rezultatai (vidutinių nuokrypių reikšmės) pateikti 2 lentelėje. Kompiuterio centrinio procesoriaus laikas vėlgi yra daugmaž vienodas visoms lygintoms mutavimo procedūroms; taigi, palyginus gautas TF santykinio nuokrypio reikšmes, matyti, kad aiškiai geresni rezultatai gaunami, kai naudojamos išsklaidyto mutavimo (M12) ir randomizuotos dalinės lokalios paieškos mutavimo (M11) procedūros. Reikia priminti, kad išsklaidyto mutavimo procedūra tiesiog remiasi randomizuota (tikimybine) tabu paieška, kombinuojant perėjimus tarp neuždraustų geriausių sprendinių ir „antrųjų“ geriausių sprendinių. Pastebėtina, jog ir paprasta mutavimo procedūra (M1), besiremianti tik atsitiktiniais poriniais elementų sukeitimais, įgalina gauti santykinai geros kokybės sprendinius.

2 lentelė. Mutavimo procedūrų palyginimo rezultatai

	Vidutinis santykinis procentinis nuokrypis											
Mut. var.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bl049	0,730	0,800	0,994	0,712	0,739	0,642	1,099	0,976	1,020	1,082	0,624	0,792
bl064	1,009	1,075	1,336	0,855	0,862	1,049	1,229	1,376	1,229	2,057	0,755	1,336
ci049	0,004	0,004	0,080	0,021	0,000	0,001	0,097	0,003	0,003	0,000	0,001	0,001
ci064	0,092	0,085	0,218	0,089	0,074	0,092	0,187	0,256	0,110	0,083	0,075	0,049
dre042	8,770	11,204	22,984	1,466	15,026	7,016	25,916	20,524	14,346	16,335	8,063	0,000
dre056	28,785	32,431	40,829	26,538	29,705	37,459	41,197	39,576	34,494	56,814	26,206	16,777
lipa070a	0,165	0,165	0,165	0,165	0,165	0,165	0,165	0,165	0,165	0,165	0,165	0,165
lipa070b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
sko056	0,002	0,000	0,082	0,019	0,000	0,000	0,096	0,064	0,003	0,001	0,000	0,000
sko064	0,006	0,000	0,002	0,006	0,000	0,000	0,007	0,026	0,001	0,000	0,006	0,000
tai035a	0,355	0,386	0,707	0,377	0,240	0,365	0,672	0,520	0,513	0,000	0,197	0,034
tai035b	0,000	0,000	0,000	0,084	0,000	0,000	0,000	0,000	0,037	0,000	0,000	0,028
tai040a	0,483	0,501	0,797	0,508	0,487	0,520	0,771	0,652	0,699	0,337	0,448	0,289
tai040b	0,000	0,201	0,000	0,000	0,000	0,000	0,000	0,201	0,402	0,000	0,000	0,603
tai045e1	3,253	1,376	10,231	11,784	2,714	2,246	9,454	8,456	17,034	0,000	1,301	15,490
tai050a	0,810	0,718	1,193	0,876	0,813	0,846	1,064	1,044	0,899	0,601	0,737	0,487
tai050b	0,000	0,000	0,078	0,253	0,000	0,033	0,019	0,033	0,035	0,000	0,033	0,123
tai060a	0,819	0,879	1,123	0,908	0,883	0,882	1,200	1,041	0,935	0,649	0,830	0,487
tai060b	0,037	0,000	0,002	0,201	0,000	0,000	0,037	0,005	0,000	0,000	0,000	0,409
wil050	0,002	0,005	0,017	0,018	0,003	0,003	0,025	0,011	0,014	0,000	0,007	0,000
Vidurkis:	2,266	2,492	4,042	2,244	2,586	2,566	4,162	3,746	3,597	3,906	1,972	1,854

Pastaba. Visais mutavimo atvejais taikytas vieno taško kryžminimo variantas.

4.3 Išplėstiniai eksperimentai su modifikuotomis parametrų reikšmėmis

Buvo taip pat atlikti papildomi, išplėstiniai eksperimentai, norint išsiaiškinti atskirų algoritmo variantų / konfigūracijų (įskaitant valdančiuosius parametrus) įtaką algoritmo veikimo efektyvumui. Iš pradžių tirtos skirtingos pradinės populiacijos konstravimo konfigūracijos. Konkrečios konfigūracijos skiriasi tuo, koks yra pradinės populiacijos dydis (narių skaičius) (PD') ir koks yra tabu paieškos algoritmo iteracijų skaičius (τ'), naudojamas sudarant pradinę populiaciją. Atskiros konfigūracijos yra tokios: 1) $PPGV = 1,\ PD' = 10,\ \tau' = 20;$ 2) $PPGV = 1,\ PD' = 20,\ \tau' = 40;$ 3) $PPGV = 1,\ PD' = 40, \tau' = 80;$ 4) $PPGV = 1,\ PD' = 100, \tau' = 200;$ 5) $PPGV = 2,\ PD' = 10, \tau' = 20;$ 6) $PPGV = 2,\ PD' = 20, \tau' = 40;$ 7) $PPGV = 2,\ PD' = 40,\ \tau' = 80;$ 8) $PPGV = 2,\ PD' = 100,\ \tau' = 200;$ 9) $PPGV = 3,\ PD' = 10,\ \tau' = 20;$ 10) $PPGV = 3,\ \ PD' = 20,\ \ \tau' = 40;$ 11) $PPGV = 3,\ \ PD' = 40,\ \ \tau' = 80;$ 12) $PPGV = 3,\ PD' = 100,\ \tau' = 200.$ Atskirų konfigūracijų palyginimo rezultatai yra pateikti 3 lentelėje, iš kurios galima matyti, jog GH algoritmo gaunami rezultatai, kaip ir buvo galima spėti, gaunami geresni, kai panaudojamas pavaldusis genetinis algoritmas ($PPGV = 2, PPGV = 3$). Taip pat matyti, jog rezultatai gerėja didinant pradinės populiacijos individų skaičių ir / arba TP algoritmo iteracijų skaičių, naudojamą pradinės populiacijos sudarymui. Išvada yra tokia, kad pradinės populiacijos kokybė reikšmingai įtakoja bendrą genetinio algoritmo efektyvumą (nežiūrint to, kad pradinės populiacijos sudarymui reikalingas papildomas skaičiavimų laikas).

Toliau buvo tirta sprendinių-tėvų atrinkimo kryžminimui variantų, t. y. konfigūracijų įtaka bendram GH efektyvumui. Variantai skiriasi tuo, kokia yra sprendinių-tėvų atrinkimo koeficiento σ reikšmė, kuri kinta nuo 1,0 iki 2,0 (žr. 4 lentelę). Pagal 4 lentelėje pateiktus rezultatus sunku nustatyti tinkamiausią atrinkimo koeficiento σ reikšmę. Gali būti, jog santykinai geriausias GHA produktyvumas pasiekiamas, kai koeficientas σ yra maždaug tarp 1,0 ir 1,3 (kas reiškia, jog tėvai kryžminimui atrenkami gana atsitiktiniu būdu).

Tolesniuose eksperimentuose bandyta nustatyti, kokia populiacijos atnaujinimo konfigūracija galėtų būti tinkamiausia. Konfigūracijų palyginimo rezultatai parodyti 5 lentelėje. Buvo tirtos tokios atskiros konfigūracijos: 1) $PAV = 1,\ PD' = 10,\ \tau' = 20;$ 2) $PAV = 2,\ PD' = 10,\ \tau' = 20;$ 3) $PAV = 3,\ PD' = 10,\ \tau' = 20;$ 4) $PAV = 1,\ PD' = 20,\ \tau' = 40;$ 5) $PAV = 2,\ PD' = 20,\ \tau' = 40;$ 6) $PAV = 3,\ PD' = 20,\ \tau' = 40;$ 7) $PAV = 1,\ PD' = 40,\ \tau' = 80;$ 8) $PAV = 2,\ PD' = 40,\ \tau' = 80;$ 9) $PAV = 3,\ PD' = 40,\ \tau' = 80;$ 10) $PAV = 1,\ PD' = 100,\ \tau' = 200;$ 11) $PAV = 2,\ PD' = 100,\ \tau' = 200;$ 12) $PAV = 3,\ \ PD' = 100,\ \ \tau' = 200$. (Čia PAV žymi populiacijos atnaujinimo variantą, PD' – populiacijos dydį pradinės populiacijos konstravimo metu, τ' – tabu paieškos procedūros iteracijų skaičių pradinės populiacijos konstravimo metu.)

Preliminari išvada yra ta, jog nėra akivaizdžios priklausomybės tarp populiacijos atnaujinimo variantų ir gaunamų sprendinių kokybės. Šiek tiek geresni rezultatai gaunami taikant antrąjį ($PAV = 2$) atnaujinimo variantą, kuris yra gana „agresyvus“, nes gautas naujas palikuonis pakeičia geriausią populiacijos narį, jeigu

tas palikuonis yra geresnis už geriausią narį. Reikalingi papildomi konfigūracijų eksperimentiniai tyrimai, norint nustatyti, ar tikrai egzistuoja kokia nors aiški priklausomybė tarp populiacijos atnaujinimo variantų ir algoritmo efektyvumo.

Toliau vykdant eksperimentus, buvo išbandytos įvairios populiacijos perkrovimo konfigūracijos. Buvo tirtos tokios konfigūracijos: 1) $PPV = 1$, $PPGV = 1$, $KV =$ "UKP", $MV =$ "M12"; 2) $PPV = 2$, $PPGV = 1$, $KV =$ "UKP", $MV =$ "M12"; 3) $PPV = 3$, $PPGV = 1$, $KV =$ "UKP",, $MV =$ "M12"; 4) $PPV = 4$, $PPGV = 1$, $KV =$ "UKP", $MV =$ "M12"; 5) $PPV = 5$, $PPGV = 1$, $KV =$ "UKP", $MV =$ "M12"; 6) $PPV = 6$, $PPGV = 1$, $KV =$ "UKP", $MV =$ "M12"; 7) $PPV = 7$, $PPGV = 1$, $KV =$ "UKP", $MV =$ "M12"; 8) $PPV = 8$, $PPGV = 1$, $KV =$ "UKP", $MV =$ "M12"; 9) $PPV = 7$, $PPGV = 2$, $KV =$ "UKP", $MV =$ "M12"; 10) $PPV = 8$, $PPGV = 2$, $KV =$ "UKP", $MV =$ "M12"; 11) $PPV = 7$, $PPGV = 3$, $KV =$ "UKP", $MV =$ "M12"; 12) $PPV = 8$, $PPGV = 3$, $KV =$ "UKP", $MV =$ "M12". (Čia PPV reiškia populiacijos perkrovimo variantą.) Šių konfigūracijų palyginimo rezultatai pateikti 6 lentelėje. Galima preliminariai teigti, jog perkrovimo variantai $(5, 7)$ yra pranašesni už kitus perkrovimo variantus, nors tai ir reikalauja daugiau kompiuterinių skaičiavimų laiko.

Atliekant tolesnius skaičiavimus, analizuotos įvairios papildomos GH algoritmo konfigūracijos, gautos skirtingais būdais kombinuojant hierarchinės iteratyviosios tabu paieškos iteracijų skaičių, taip pat hierarchinių lygių skaičių. Išbandyti šie HITP algoritmo veikimo scenarijai: 1) „greitos paieškos“ scenarijus: maža τ reikšmė – maža Q_{hier} reikšmė; 2) „diversifikuotos greitos paieškos“ scenarijus: maža τ reikšmė – didelė Q_{hier} reikšmė; 3) „intensyvios paieškos“ scenarijus: didelė τ reikšmė – maža Q_{hier} reikšmė; 4) „diversifikuotos intensyvios paieškos“ scenarijus: didelė τ reikšmė – didelė Q_{hier} reikšmė. Šiuose scenarijuose genetinio algoritmo generacijų skaičius turi būti subalansuojamas, t. y. sumažinamas siekiant, kad algoritmo vykdymo bendras laikas būtų iš esmės nepakitęs. Tokiu būdu algoritmo valdymo parametrų reikšmės skirtinguose scenarijuose yra šios: 1-a) $\tau = 20$, $Q_{hier} = 2^8 = 256$, $N_{gen} = 100$; 1-b) $\tau = 25$, $Q_{hier} = 2^8 = 256$, $N_{gen} = 80$; 1-c) $\tau = 40$, $Q_{hier} = 2^8 = 256$, $N_{gen} = 50$; 2-a) $\tau = 50$, $Q_{hier} = 2^8 = 256$, $N_{gen} = 40$; 2-b) $\tau = 10$, $Q_{hier} = 2^9 = 512$, $N_{gen} = 100$; 2-c) $\tau = 20$, $Q_{hier} = 2^9 = 512$, $N_{gen} = 50$; 3-a) $\tau = 10$, $Q_{hier} = 5 \times 2^7 = 640$, $N_{gen} = 80$; 3-b) $\tau = 20$, $Q_{hier} = 5 \times 2^7 = 640$, $N_{gen} = 40$; 3-c) $\tau = 10$, $Q_{hier} = 2^{10} = 1024$, $N_{gen} = 50$; 4-a) $\tau = 20$, $Q_{hier} = 2^{10} = 1024$, $N_{gen} = 25$; 4-b) $\tau = 10$, $Q_{hier} = 10 \times 2^7 = 1280$, $N_{gen} = 40$; 4-c) $\tau = 20$, $Q_{hier} = 10 \times 2^7 = 1280$, $N_{gen} = 20$.

Gauti eksperimentų rezultatai, pateikti 7 lentelėje, liudija, jog „diversifikuotos intensyvios paieškos“ scenarijus gana akivaizdžiai dominuoja kitų konfigūracinių variantų, t. y. scenarijų, atžvilgiu. Dar didesnio rezultatų kokybės pagerėjimo galima tikėtis, jau tik panaudojant „ekstensyvius“ būdus, t. y. padidinant arba genetinio algoritmo generacijų skaičių, arba TP/HITP algoritmo iteracijų skaičių, arba pagaliau pačių hierarchinių HITP algoritmo lygių kiekį.

Baigiamuosiuose kompiuteriniuose skaičiavimuose ir buvo tirta, kaip galimai pagerėja sprendinių kokybė, padidinant vidinės tabu paieškos procedūros bendrą

iteracijų skaičių (τ) arba HITP algoritmo iteracijų skaičių Q_{hier}. Atliktų tyrimų rezultatai pateikti 8 lentelėje. Atskiruose algoritmo variantuose buvo naudoti tokie parametrai: 1) $PD = 10,\ PD' = 150,\ \tau' = 300,\ Q_{hier} = 5 \times 2^7 = 640$; 2) $PD = 10,\ PD' = 150,\ \tau' = 300,\ Q_{hier} = 6 \times 2^7 = 768$; 3) $PD = 10,\ PD' = 150,\ \tau' = 300,\ Q_{hier} = 7 \times 2^7 = 896$; 4) $PD = 10,\ PD' = 150,\ \tau' = 300,\ Q_{hier} = 8 \times 2^7 = 1024$; 5) $PD = 20,\ PD' = 300,\ \tau' = 300,\ Q_{hier} = 5 \times 2^7 = 640$; 6) $PD = 20,\ PD' = 300,\ \tau' = 300,\ Q_{hier} = 6 \times 2^7 = 768$; 7) $PD = 20,\ PD' = 300,\ \tau' = 300,\ Q_{hier} = 7 \times 2^7 = 896$; 8) $PD = 20,\ PD' = 300,\ \tau' = 300,\ Q_{hier} = 8 \times 2^7 = 1024$; 9) $PD = 10,\ PD' = 200,\ \tau' = 400,\ Q_{hier} = 7 \times 2^7 = 896$; 10) $PD = 10,\ PD' = 200,\ \tau' = 400,\ Q_{hier} = 7 \times 2^7 = 1024$; 11) $PD = 20,\ PD' = 400,\ \tau' = 400,\ Q_{hier} = 7 \times 2^7 = 896$; 12) $PD = 20,\ PD' = 400,\ \tau' = 400,\ Q_{hier} = 7 \times 2^7 = 1024$. Priminsime, kad PD' žymi pradinės populiacijos dydį, o τ' yra TP algoritmo iteracijų skaičius, naudojamas pradinei populiacijai sudaryti.

Akivaizdu, jog yra žymus gaunamų rezultatų kokybės pagerėjimas, esant padidintam vykdomų HITP iteracijų skaičiui. Tai aiškiai liudija hierarchinio principo tikslingumą ir naudingumą (žr. 21 pav.). Tas gali būti taip pat matoma ir, taip sakant, žvelgiant iš kitos „perspektyvos“. Tuo tikslu buvo sudarytos tikslo funkcijos reikšmių dažnių diagramos vienam iš sudėtingų testinių duomenų pavyzdžių – `bl064`. Konkrečiai kalbant, buvo sudarytos vidutinio santykinio nuokrypio ($\bar{\theta}$) dažnių (tarp 0 ir 10) diagramos, kuomet algoritmo vykdymų skaičius yra lygus 10 (žr. 22 pav.). Diagramose horizontalioje ašyje esanti reikšmė 0,0 reiškia nulinį nuokrypį, o reikšmė 1,0 atitinka maksimalų galimą nuokrypį. Diagramos sudarytos tokiu būdu, jog vidutinio nuokrypio ($\bar{\theta}$) dažniai yra vizualiai pateikti dešimtyje dalinių intervalų: [0,0; 0,1); [0,1; 0,2); [0,2; 0,3); ...; [0,9; 1,0). Iš diagramų matyti, kad vidutinis tikslo funkcijos nuokrypis stabiliai mažėja didinant iteracijų skaičių Q_{hier}.

Vis dėlto turi būti pastebėta, jog reikėtų daugiau eksperimentų, siekiant nustatyti galimą optimalų hierarchinių lygių skaičių, tuo pačiu geriausią balansą tarp TP iteracijų skaičiaus ir HITP algoritmo hierarchinių lygmenų kiekio.

3 lentelė. Pradinės populiacijos konstravimo variantų palyginimo rezultatai

	Vidutinis santykinis procentinis nuokrypis											
Konstr. var.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bl049	0,607	0,589	0,554	0,607	0,668	0,598	0,589	0,624	0,616	0,598	0,519	0,493
bl064	0,601	0,835	0,741	0,661	0,741	0,501	0,768	0,681	0,735	0,681	0,715	0,661
ci049	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
ci064	0,055	0,019	0,029	0,000	0,051	0,060	0,028	0,000	0,008	0,007	0,000	0,000
dre042	4,267	3,272	6,466	0,000	6,309	4,869	4,293	0,000	1,335	0,000	0,000	0,000
dre056	23,462	20,626	12,302	4,494	13,131	22,118	13,941	11,013	20,166	19,153	4,807	3,757
lipa070a	0,055	0,055	0,055	0,055	0,055	0,055	0,055	0,055	0,055	0,055	0,055	0,055
lipa070b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
sko056	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
sko064	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai035a	0,201	0,169	0,076	0,000	0,127	0,081	0,000	0,000	0,000	0,000	0,000	0,000
tai035b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai040a	0,443	0,377	0,335	0,219	0,512	0,277	0,263	0,083	0,311	0,231	0,088	0,067
tai040b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai045e1	0,730	0,730	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai050a	0,647	0,715	0,628	0,450	0,620	0,610	56,000	0,352	0,577	0,488	0,372	0,191
tai050b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai060a	0,721	0,672	0,643	0,519	0,667	0,660	0,549	0,463	0,633	0,506	0,460	0,353
tai060b	0,037	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
wil050	0,003	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Vidurkis:	1,591	1,403	1,091	0,350	1,144	1,491	3,824	0,664	1,222	1,086	0,351	0,279

Pastaba. Visais atvejais taikytas universaliojo kryžminimo variantas (UKP) ir dvyliktas mutavimo variantas (M12).

4 lentelė. Atrinkimo variantų palyginimo rezultatai

	Vidutinis santykinis procentinis nuokrypis											
σ	**1,0**	**1,1**	**1,2**	**1,3**	**1,4**	**1,5**	**1,6**	**1,7**	**1,8**	**1,9**	**1,95**	**2,0**
bl049	0,730	0,774	0,730	0,668	0,642	0,633	0,695	0,633	0,730	0,748	0,712	0,792
bl064	1,009	0,835	0,935	0,942	0,842	0,982	1,029	0,955	0,768	0,895	0,902	0,969
ci049	0,004	0,001	0,009	0,001	0,001	0,012	0,009	0,000	0,000	0,000	0,003	0,003
ci064	0,092	0,065	0,074	0,085	0,073	0,081	0,089	0,087	0,079	0,067	0,111	0,085
dre042	8,770	10,942	7,382	9,058	15,026	10,105	10,209	19,686	6,387	3,717	7,775	10,000
dre056	28,785	28,637	34,494	28,582	28,619	33,444	32,228	33,812	23,665	31,344	33,812	29,429
lipa070a	0,165	0,165	0,165	0,165	0,165	0,165	0,165	0,165	0,165	0,165	0,165	0,165
lipa070b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
sko056	0,002	0,001	0,001	0,001	0,001	0,000	0,000	0,038	0,001	0,001	0,000	0,001
sko064	0,006	0,000	0,006	0,000	0,006	0,000	0,000	0,012	0,000	0,000	0,000	0,022
tai035a	0,355	0,333	0,368	0,294	0,420	0,388	0,401	0,304	0,408	0,304	0,261	0,248
tai035b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai040a	0,483	0,545	0,586	0,458	0,534	0,475	0,608	0,545	0,503	0,504	0,578	0,550
tai040b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai045e1	3,253	2,236	1,192	1,344	1,971	1,843	1,037	1,382	0,895	38,886	2,143	2,074
tai050a	0,810	0,813	0,833	0,835	0,787	0,858	0,765	0,786	0,762	0,732	0,900	0,756
tai050b	0,000	0,000	0,000	0,033	0,000	0,033	0,033	0,033	0,000	0,033	0,000	0,033
tai060a	0,819	0,820	0,838	0,795	0,786	0,809	0,808	0,877	0,845	0,879	0,879	0,987
tai060b	0,037	0,000	0,037	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,037
wil050	0,002	0,000	0,005	0,003	0,007	0,005	0,005	0,005	0,007	0,000	0,002	0,003
Vidurkis:	2,266	2,308	2,383	2,163	2,494	2,492	2,404	2,966	1,761	3,914	2,412	2,308

Pastabos. σ yra sprendinių-tėvų atrinkimo koeficientas (žr. (27) formulę). Visais atvejais taikyta kryžminimo procedūra UKP ir mutavimo procedūra M12.

5 lentelė. Populiacijos atnaujinimo variantų palyginimo rezultatai

	Vidutinis santykinis procentinis nuokrypis											
Atnauj. var.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bl049	0,607	0,624	0,677	0,589	0,642	0,580	0,554	0,624	0,616	0,607	0,589	0,589
bl064	0,601	0,635	0,715	0,835	0,715	0,688	0,741	0,695	0,768	0,661	0,635	0,755
ci049	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
ci064	0,055	0,035	0,040	0,019	0,051	0,036	0,029	0,032	0,027	0,000	0,000	0,000
dre042	4,267	6,963	8,246	3,272	1,492	2,094	6,466	1,466	5,419	0,000	0,000	0,000
dre056	23,462	15,488	9,687	20,626	11,326	18,250	12,302	8,122	10,055	4,494	5,783	7,035
lipa070a	0,055	0,055	0,055	0,055	0,055	0,055	0,055	0,055	0,055	0,055	0,055	0,055
lipa070b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
sko056	0,000	0,001	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
sko064	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai035a	0,201	0,222	0,142	0,169	0,130	0,165	0,076	0,076	0,032	0,000	0,000	0,000
tai035b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai040a	0,443	0,410	0,444	0,377	0,415	0,438	0,335	0,326	0,296	0,219	0,219	0,222
tai040b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai045e1	0,730	2,920	2,492	0,730	1,023	0,605	0,000	0,000	0,459	0,000	0,000	0,000
tai050a	0,647	0,701	0,648	0,715	0,671	0,685	0,628	0,611	0,606	0,450	0,450	0,424
tai050b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai060a	0,721	0,740	0,687	0,672	0,690	0,744	0,643	0,627	0,627	0,519	0,469	0,501
tai060b	0,037	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
wil050	0,003	0,002	0,003	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Vidurkis:	**1,591**	**1,440**	**1,192**	**1,403**	**0,861**	**1,217**	**1,091**	**0,632**	**0,948**	**0,350**	**0,410**	**0,479**

Pastabos. Visais atvejais taikyta kryžminimo procedūra UKP ir mutavimo procedūra M12. Taip pat naudotas pradinės populiacijos generavimo variantas $PPGV = 1$.

6 lentelė. Populiacijos perkrovimo variantų palyginimo rezultatai

	Vidutinis santykinis procentinis nuokrypis											
Perkr. var.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bl049	0,616	0,493	0,572	0,493	0,554	0,493	0,642	0,572	0,642	0,686	0,607	0,580
bl064	0,708	0,755	0,728	0,768	0,701	0,701	0,708	0,762	0,695	0,848	0,728	0,715
ci049	0,001	0,000	0,000	0,001	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
ci064	0,035	0,015	0,057	0,059	0,037	0,029	0,012	0,033	0,013	0,031	0,018	0,000
dre042	5,942	1,859	8,639	3,325	1,859	6,414	3,796	3,351	7,330	5,052	0,000	3,534
dre056	26,206	20,810	22,891	28,858	19,227	21,087	24,273	20,810	21,013	22,413	19,540	23,241
lipa070a	0,052	0,052	0,052	0,052	0,052	0,052	0,052	0,052	0,052	0,052	0,052	0,052
lipa070b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
sko056	0,001	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
sko064	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai035a	0,338	0,166	0,184	0,104	0,096	0,158	0,070	0,000	0,062	0,058	0,000	0,000
tai035b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai040a	0,456	0,286	0,455	0,349	0,401	0,371	0,271	0,242	0,314	0,338	0,239	0,212
tai040b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai045e1	1,023	0,459	0,730	1,335	0,730	2,190	0,000	0,000	0,000	0,000	0,000	0,165
tai050a	0,764	0,568	0,647	0,600	0,633	0,567	0,638	0,654	0,663	0,544	0,505	0,512
tai050b	0,033	0,000	0,033	0,033	0,033	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai060a	0,676	0,695	0,649	0,698	0,677	0,582	0,663	0,634	0,612	0,577	0,587	0,642
tai060b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
wil050	0,007	0,000	0,003	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Vidurkis:	1,843	1,308	1,782	1,834	1,250	1,632	1,556	1,356	1,570	1,530	1,114	1,483

Pastabos. Visais atvejais taikyta kryžminimo procedūra UKP ir mutavimo procedūra M12. Taip pat naudotas pradinės populiacijos generavimo variantas $PPGV = 1$.

7 lentelė. Hierarchinės iteratyviosios tabu paieškos algoritmo konfigūracijų palyginimo rezultatai

Vidutinis santykinis procentinis nuokrypis												
Konfig. Nr.	**1 a)**	**1 b)**	**1 c)**	**2 a)**	**2 b)**	**2 c)**	**3 a)**	**3 b)**	**3 c)**	**4 a)**	**4 b)**	**4 c)**
bl049	0,598	0,545	0,633	0,633	0,440	0,589	0,528	0,475	0,528	0,510	0,475	0,510
bl064	0,735	0,675	0,788	0,935	0,721	0,715	0,661	0,808	0,695	0,681	0,675	0,755
ci049	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
ci064	0,031	0,014	0,019	0,021	0,011	0,017	0,004	0,011	0,007	0,000	0,004	0,000
dre042	3,613	1,466	0,000	2,670	0,000	1,335	0,000	0,000	0,000	0,000	0,000	0,000
dre056	8,692	13,297	14,512	16,133	17,017	19,227	13,223	7,974	9,816	5,506	6,851	5,672
sko049	0,052	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
sko056	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
sko064	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai020a	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai035a	0,108	0,136	0,000	0,000	0,077	0,020	0,063	0,078	0,000	0,000	0,067	0,000
tai035b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai040a	0,342	0,337	0,350	0,277	0,362	0,264	0,315	0,275	0,322	0,263	0,267	0,201
tai040b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai045e1	0,586	0,730	0,293	2,015	0,000	0,293	0,000	0,000	0,000	0,000	0,000	0,000
tai050a	0,487	0,599	0,559	0,499	0,613	0,612	0,614	0,527	0,504	0,524	0,481	0,491
tai050b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai060a	0,649	0,607	0,620	0,582	0,573	0,564	0,562	0,635	0,543	0,584	0,542	0,511
tai060b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
wil050	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Vidurkis:	**0,795**	**0,920**	**0,889**	**1,188**	**0,991**	**1,182**	**0,799**	**0,539**	**0,621**	**0,403**	**0,468**	**0,407**

Pastabos. Visais atvejais taikyta kryžminimo procedūra UKP ir mutavimo procedūra M12. Taip pat naudotas pradinės populiacijos generavimo variantas $PPGV = 1$ bei populiacijos atnaujinimo variantas $PAV = 2$.

8 lentelė. Papildomų GH algoritmo konfigūracijų su padidintu HITP iteracijų skaičiumi palyginimo rezultatai

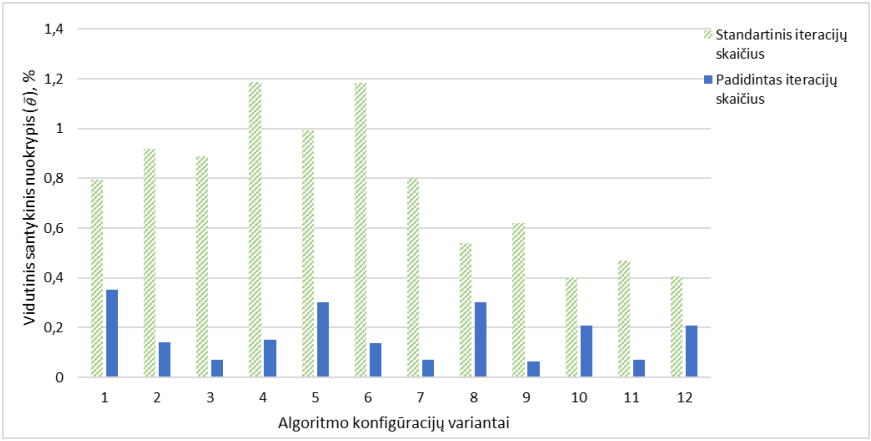
	Vidutinis santykinis procentinis nuokrypis											
Konfig. Nr.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bl049	0,440	0,466	0,466	0,396	0,475	0,484	0,396	0,440	0,484	0,396	0,484	0,466
bl064	0,615	0,608	0,568	0,541	0,548	0,528	0,574	0,635	0,508	0,521	0,648	0,581
ci049	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
ci064	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
dre042	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
dre056	5,654	1,344	0,000	1,731	4,696	1,289	0,000	4,678	0,000	2,910	0,000	2,910
sko049	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
sko056	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
sko064	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai020a	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai035a	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai035b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai040a	0,015	0,030	0,037	0,037	0,022	0,037	0,030	0,037	0,037	0,037	0,022	0,022
tai040b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai045e1	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai050a	0,084	0,116	0,117	0,080	0,092	0,152	0,119	0,057	0,036	0,092	0,052	0,011
tai050b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
tai060a	0,221	0,258	0,237	0,238	0,235	0,236	0,277	0,221	0,214	0,211	0,200	0,161
tai060b	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
wil050	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Vidurkis:	**0,351**	**0,141**	**0,071**	**0,151**	**0,303**	**0,136**	**0,070**	**0,303**	**0,064**	**0,208**	**0,070**	**0,208**

Pastabos. Visais atvejais taikyta kryžminimo procedūra UKP ir mutavimo procedūra M12. Panaudotas pradinės populiacijos generavimo variantas $PPGV = 3$ bei populiacijos atnaujinimo variantas $PAV = 2$.

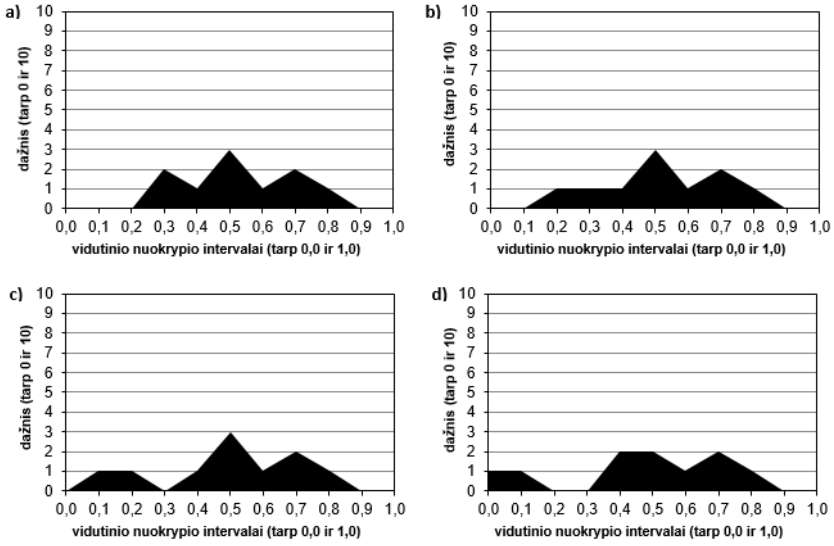
9 lentelė. Gautų geriausių žinomų ((pseudo)optimalių) sprendinių skaičius

Geriausių žinomų sprendinių skaičius												
Konfig. Nr.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bl049	0	0	0	1	0	0	0	0	0	0	0	0
bl064	0	0	0	0	0	1	0	0	0	0	0	0
ci049	10	10	10	10	10	10	10	10	10	10	10	10
ci064	10	10	10	10	10	10	10	10	10	10	10	10
dre042	10	10	10	10	10	10	10	10	10	10	10	10
dre056	7	9	10	9	7	9	10	7	10	8	10	8
lipa070a	10	10	10	10	10	10	10	10	10	10	10	10
lipa070b	10	10	10	10	10	10	10	10	10	10	10	10
sko056	10	10	10	10	10	10	10	10	10	10	10	10
sko064	10	10	10	10	10	10	10	10	10	10	10	10
tai035a	10	10	10	10	10	10	10	10	10	10	10	10
tai035b	10	10	10	10	10	10	10	10	10	10	10	10
tai040a	8	6	5	5	7	5	6	5	5	5	7	7
tai040b	10	10	10	10	10	10	10	10	10	10	10	10
tai045e1	10	10	10	10	10	10	10	10	10	10	10	10
tai050a	3	3	3	6	5	2	2	3	6	1	4	8
tai050b	10	10	10	10	10	10	10	10	10	10	10	10
tai060a	1	0	1	0	0	0	0	0	0	0	0	3
tai060b	10	10	10	10	10	10	10	10	10	10	10	10
wil050	10	10	10	10	10	10	10	10	10	10	10	10
Suma:	159	158	159	161	159	157	158	155	161	154	161	166

Pastaba. Gautų geriausių žinomų ((pseudo)optimalių) sprendinių skaičiai pateikti tiems patiems eksperimentams, kaip ir 8 lentelėje.



21 pav. GH algoritmo konfigūracijų su standartiniu ir padidintu iteracijų skaičiumi palyginimas



22 pav. Duomenų pavyzdžio `bl064` gautų tikslo funkcijos reikšmių dažnių diagramos skirtingiems tirtiems scenarijams: a) $\boldsymbol{PD = 10}$, $\boldsymbol{PD' = 150}$, $\boldsymbol{\tau' = 300}$, $\boldsymbol{Q_{hier} = 640}$; b) $\boldsymbol{PD = 10}$, $\boldsymbol{PD' = 150}$, $\boldsymbol{\tau' = 300}$, $\boldsymbol{Q_{hier} = 768}$; c) $\boldsymbol{PD = 10}$, $\boldsymbol{PD' = 150}$, $\boldsymbol{\tau' = 300}$, $\boldsymbol{Q_{hier} = 896}$; d) $\boldsymbol{PD = 10}$, $\boldsymbol{PD' = 150}$, $\boldsymbol{\tau' = 300}$, $\boldsymbol{Q_{hier} = 1024}$

Galima aiškiai pastebėti, jog GH algoritmas pasižymi išties akivaizdžiomis potencinėmis galimybėmis tuo požiūriu, jog gaunamų sprendinių kokybė (išreiškiama būtent tikslo funkcijos vidutiniu nuokrypiu nuo optimumo) pastebimai pagerėja „apmokius“ algoritmą, kitaip tariant, suformavus tinkamą algoritmo komponentų ir valdančiųjų parametrų konfigūraciją. Kaip to konkretų patvirtinimą galima nurodyti iliustratyvius eksperimentų rezultatus, pateiktus 1 lentelės pirmame stulpelyje ir 8 lentelės devintame stulpelyje (tai yra tik keletas reprezentatyvių rezultatų, bet jų būtų

galima nurodyti ir daugiau). Taigi, 1 lentelės pirmame stulpelyje yra pateikti „bazinio“ algoritmo su standartiniais komponentais (kryžminimu, mutavimu) ir nekalibruotais parametrais rezultatai. Tuo tarpu 8 lentelės devintame stulpelyje matomi sukalibruoto algoritmo rezultatai. Matyti, kad, pvz., testinio duomenų pavyzdžio `dre056` rezultatų pagerėjimas absoliučia išraiška siekia 28,785 (procentine išraiška – 100%); testinio pavyzdžio `dre042` pagerėjimas yra lygus 8,77 (100%); testinio pavyzdžio `tai045e1` pagerėjimas yra 3,253 (100%); testinio pavyzdžio `tai050a` – 0,774 (96%); testinio pavyzdžio `tai060a` – 0,605 (74%); testinio pavyzdžio `bl064` – 0,501 (50%), ir t. t. (žr. 10 lentelę).

10 lentelė. Algoritmų rezultatų kiekybinio pagerėjimo iliustravimas

Testiniai duomenys	**Algoritmo rezultatų pagerėjimas**					
	„1 scenarijus“		**„2 scenarijus“**		**„3 scenarijus“**	
	κ	υ	κ	υ	κ	υ
bl049	0,246	33,699	**0,281**	**36,732**	0,228	32,022
bl064	**0,501**	**49,653**	0,267	34,452	0,260	33,854
dre042	8,770	**100,000**	**15,052**	**100,000**	13,665	**100,000**
dre056	28,785	**100,000**	28,674	**100,000**	**36,206**	**100,000**
tai035a	0,355	**100,000**	**0,416**	**100,000**	0,332	**100,000**
tai040a	0,446	92,340	0,380	91,127	**0,519**	**93,345**
tai045e1	**3,253**	**100,000**	1,366	**100,000**	2,944	**100,000**
tai050a	0,774	95,556	0,798	95,683	**0,836**	**95,872**
tai060a	0,605	73,871	0,612	74,092	**0,690**	**76,327**
Vidurkis:	4,390	84,512	4,801	83,209	5,585	83,142

Pastabos. 1. Algoritmo gerinimo scenarijai („trajektorijos“) „1 scenarijus“, „2 scenarijus“, „3 scenarijus“ skiriasi vienas nuo kito priklausomai nuo to, koks pradinis algoritmo variantas naudojamas (žr. 1 lentelės 1, 2, 3 stulpelius). 2. κ žymi absoliutinį pagerėjimą, o υ – procentinį pagerėjimą; čia $\kappa = \bar{\theta} - \bar{\theta}'$, $\bar{\theta}$ yra pradinis TF nuokrypis (žr. 1 lentelės 1, 2, 3 stulpelius), $\bar{\theta}'$ – TF nuokrypis po pagerėjimo (žr. 8 lentelės 9 stulpelį); $\upsilon = \frac{\bar{\theta} - \bar{\theta}'}{\bar{\theta}} \times 100[\%]$.

Įdomu pastebėti, kad sunkiai sprendžiamų testinių pavyzdžių `dre042`, `dre056` TF nuokrypis buvo sumažintas daugiausiai (absoliutinis skirtumas yra atitinkamai 15,052 ir 36,206; tuo tarpu mažiausias pagerinimas buvo gautas kitų sunkių testinių pavyzdžių `bl049` ir `bl064` atvejais (žr. 10 lentelę)).

Galima pateikti tokius apibendrintus statistinius rezultatus. 20 tirtų testinių duomenų failų geriausi žinomi, t. y. (pseudo-)optimalūs sprendiniai buvo gauti 12016 kartų iš 19200 algoritmo vykdymų (daugiau kaip 62 % visų atvejų). Bendras vidutinis gautų tikslo funkcijos reikšmių nuokrypis yra lygus 1,59 %. Vidutinis algoritmo vykdymo laikas lygus apytiksliai 60 s. Vidutinis nuokrypis mažesnis negu 0,5 yra 71 % visų atvejų; tuo tarpu vidutinis nuokrypis mažesnis negu 1,0 yra 88 % visų atvejų. 14 testinių duomenų pavyzdžių (`ci049, ci064, dre042,`

lipa070a, lipa070b, sko056, sko064, tai035a, tai035b, tai040b, tai045e1, tai050b, tai060b, wil050) buvo išspręsti (pseudo-)optimaliai (t. y. buvo gauti (pseudo-)optimalūs sprendiniai) daugiau negu 200 kartų. Be to, 8 lentelėje galima matyti, kad atliekant eksperimentus, kai buvo padidintas HITP iteracijų skaičius, iš 20 tirtų duomenų failų 14 failų (t. y. 50 %) vidutinis santykinis nuokrypis yra lygus 0; o tai reiškia, jog visais šiais atvejais buvo gauti (pseudo-)optimalūs sprendiniai. Taip pat 9 lentelėje galime pastebėti, kad 3 testuotiems duomenų failams (pseudo-)optimalūs sprendiniai pasiekti tarp 46 ir 104 kartų. 3 duomenų failams, kurie yra ypač sunkūs, (pseudo-)optimalūs sprendiniai gauti tarp 1 ir 5 kartų. Tačiau ir šiems duomenų pavyzdžiams būtų galima rasti daugiau (pseudo-)optimalių sprendinių, dar labiau padidinus TP ir / arba HITP algoritmų iteracijų skaičių.

Įvykdžius preliminarius eksperimentus su 20 testinių duomenų pavyzdžių, buvo taip pat atlikti eksperimentai su likusiais kitais duomenų pavyzdžiais, kurių iš viso yra 88. Eksperimentai atlikti su patikslintomis parametrų reikšmėmis, atsižvelgiant į preliminarių eksperimentų rezultatus. Rezultatai, kurie pateikti 11 lentelėje, iliustruoja, kaip greitai GH algoritmas konverguoja į (pseudo-)optimalius sprendinius. Čia matyti, jog visų 88 tirtų įvairių dydžių ($10 \leq n \leq 128$) duomenų pavyzdžių GH algoritmo su atiderintomis parametrų reikšmėmis konvergavimo į (pseudo-)optimalius sprendinius laikas yra iš ties labai mažas. Vidutinis (pseudo-)optimalių sprendinių suradimo laikas yra lygus 14,87 s.

11 lentelė. GHA rezultatai 88 duomenų failams iš QAPLIB bibliotekos (Burkard ir kt., 1997) ir straipsnių (Drezner ir kt., 2005), (Drezner, Misevičius, 2013)

Duomenų pavyzdžiai	GŽR	$\overline{\theta}$	Laikas (sek.)	Duomenų pavyzdžiai	GŽR	$\overline{\theta}$	Laikas (sek.)
bl036	3296	0,000	9,491	**lipa090b**	12490441	0,000	0,803
chr025a	3796	0,000	1,936	**nug030**	6124	0,000	0,122
ci036	168611971	0,000	1,279	**rou020**	725522	0,000	0,089
ci049	236355034	0,000	6,864	**scr020**	110030	0,000	0,022
ci064	325671035	0,000	46,818	**sko042**	15812	0,000	0,592
ci081	427447820	0,000	250,470	**sko049**	23386	0,000	7,989
dre015	306	0,000	0,003	**sko056**	34458	0,000	7,145
dre018	332	0,000	0,034	**sko064**	48498	0,000	7,833
dre021	356	0,000	0,033	**ste036a**	9526	0,000	0,830
dre024	396	0,000	0,178	**ste036b**	15852	0,000	0,175
dre028	476	0,000	0,470	**ste036c**	8239110	0,000	0,513
dre030	508	0,000	0,875	**tai010a**	135028	0,000	0,005
dre042	764	0,000	9,809	**tai010b**	1183760	0,000	0,003
dre056	1086	0,000	86,024	**tai012a**	224416	0,000	0,003
dre072	1452	0,000	489,877	**tai012b**	39464925	0,000	0,003

Pastaba. Laikas žymi vieno vykdymo vidutinį CP laiką.

11 lentelė (tęsinys). GHA rezultatai 88 duomenų failams iš QAPLIB bibliotekos (Burkard ir kt., 1997) ir straipsnių (Drezner ir kt., 2005), (Drezner, Misevičius, 2013)

Duomenų pavyzdžiai	GŽR	$\overline{\theta}$	Laikas (sek.)	Duomenų pavyzdžiai	GŽR	$\overline{\theta}$	Laikas (sek.)
els019	17212548	0,000	0,023	**tai015a**	388214	0,000	0,006
esc032a	130	0,000	0,237	**tai015b**	51765268	0,000	0,005
esc032b	168	0,000	0,022	**tai017a**	491812	0,000	0,009
esc032c	642	0,000	0,005	**tai020a**	703482	0,000	0,122
esc032d	200	0,000	0,008	**tai020b**	122455319	0,000	0,014
esc032e	2	0,000	0,003	**tai025a**	1167256	0,000	0,262
esc032f	2	0,000	0,005	**tai025b**	344355646	0,000	0,041
esc032g	6	0,000	0,003	**tai027e1**	2558	0,000	0,332
esc032h	438	0,000	0,008	**tai027e2**	2850	0,000	0,399
esc064a	116	0,000	0,026	**tai027e3**	3258	0,000	0,078
esc128	64	0,000	0,335	**tai030a**	1818146	0,000	0,392
had020	6922	0,000	0,013	**tai030b**	637117113	0,000	0,176
kra030a	88900	0,000	0,304	**tai035a**	2422002	0,000	1,527
kra030b	91420	0,000	0,643	**tai035b**	283315445	0,000	0,800

Pastaba. Laikas žymi vieno vykdymo vidutinį CP laiką.

11 lentelė (tęsinys). GHA rezultatai 88 duomenų failams iš QAPLIB bibliotekos (Burkard ir kt., 1997) ir straipsnių (Drezner ir kt., 2005), (Drezner, Misevičius, 2013)

Duomenų pavyzdžiai	GŽR	$\overline{\theta}$	Laikas (sek.)	Duomenų pavyzdžiai	GŽR	$\overline{\theta}$	Laikas (sek.)
lipa020a	3683	0,000	0,009	**tai040b**	637250948	0,000	0,900
lipa020b	27076	0,000	0,002	**tai045e1**	6412	0,000	1,346
lipa030a	13178	0,000	0,038	**tai045e2**	5734	0,000	5,713
lipa030b	151426	0,000	0,008	**tai045e3**	7438	0,000	2,471
lipa040a	31538	0,000	0,190	**tai050b**	458821517	0,000	5,488
lipa040b	476581	0,000	0,017	**tai060b**	608215054	0,000	5,036
lipa050a	62093	0,000	0,473	**tai064c**	1855928	0,000	0,022
lipa050b	1210244	0,000	0,062	**tai075e1**	14488	0,000	52,287
lipa060a	107218	0,000	4,446	**tai075e2**	14444	0,000	25,134
lipa060b	2520135	0,000	0,153	**tai075e3**	14154	0,000	36,677
lipa070a	169755	0,000	6,915	**tai080b**	818415043	0,000	29,161
lipa070b	4603200	0,000	0,251	**tai100b**	1185996137	0,000	83,515
lipa080a	253195	0,000	22,615	**tho030**	149936	0,000	0,097
lipa080b	7763962	0,000	0,579	**tho040**	240516	0,000	3,928
lipa090a	360630	0,000	81,371	**wil050**	48816	0,000	3,133

Pastaba. Laikas žymi vieno vykdymo vidutinį CP laiką.

4.4 Palyginimo eksperimentai

Buvo taip pat atlikti GH algoritmo ir kitų euristinių algoritmų palyginimo eksperimentai. Pirmiausiai GH algoritmas buvo palygintas su memetiniu algoritmu (MA) (Benlic, Hao, 2015) ir dažnų pėdsakų paieškos algoritmu (DPP) (Zhou ir kt., 2020), kurie, turimomis žiniomis, yra efektyviausi iki šiol euristiniai algoritmai KP uždaviniui. Palyginimo eksperimentų rezultatai pateikti 12, 13, 14, 15 lentelėse. Iš šių rezultatų (būtent algoritmų vykdymo laikų) matyti, kad GH algoritmas pranoksta memetinį algoritmą ir iš esmės yra panašaus efektyvumo lygio, kaip ir dažnų pėdsakų paieškos algoritmas. Algoritmų vykdymo laikų skirtumas GH algoritmo naudai atskiriems testinių duomenų pavyzdžiams siekia iki 64 kartų (žr. 14 lentelę).

Kituose eksperimentuose buvo palygintas GH algoritmas ir du kiti efektyvūs euristiniai algoritmai, būtent hibridinis genetinis algoritmas (HGA) (Drezner ir kt., 2005), bei hibridinis genetinis algoritmas su diferencijuotu pagerinimu (HGA-DI) (Drezner, Misevičius, 2013). Eksperimentų rezultatai pateikti 16, 17, 18, 19 lentelėse. Ir vėl gauti eksperimentų rezultatai liudija, jog GH algoritmas yra iš esmės pranašesnis tiek už algoritmą HGA, tiek už algoritmą HGA-DI gaunamų sprendinių kokybės atžvilgiu. Algoritmų GH ir HGA vykdymo laikų skirtumas kai kuriais atvejais siekia net 180 kartų (žr. 16 lentelę). Algoritmų GH ir HGA-DI vykdymo laikų skirtumas, tuo tarpu, siekia 39 kartus (žr. 19 lentelę).

12 lentelė. GHA ir memetinio algoritmo (MA) (Benlic, Hao, 2015) palyginimo rezultatai (I)

Duomenų pavyzdžiai	GŽR	GHA		MA	
		$\overline{\theta}$	Laikas (sek.)	$\overline{\theta}$	Laikas (sek.)
sko072	66256	**0,000**	**29,380**	**0,000**	240,000
sko081	90998	**0,000**	**95,421**	**0,000**	258,000
sko090	115534	**0,000**	**229,456**	**0,000**	918,000
sko100a	152002	**0,000**	**542,640**	**0,000**	1338,000
sko100b	153890	**0,000**	**227,774**	**0,000**	390,000
sko100c	147862	**0,000**	**400,697**	**0,000**	720,000
sko100d	149576	**0,000**	**377,108**	**0,006**	1254,000
sko100e	149150	**0,000**	**438,632**	**0,000**	714,000
sko100f	149036	**0,000**	**790,550**	**0,000**	1380,000
wil100	273038	**0,000**	**600,566**	**0,000**	870,000
tho150	8133398	**0,007**	**6300,000**	0,008	24984,000

Pastabos. 1. Laikas žymi vieno vykdymo vidutinį CP laiką. 2. Originaliame (Benlic, Hao, 2015) straipsnyje laikas yra pateiktas minutėmis.

13 lentelė. GHA ir memetinio algoritmo (MA) (Benlic, Hao, 2015) palyginimo rezultatai (II)

Duomenų pavyzdžiai	GŽR	GHA		MA	
		$\overline{\theta}$	Laikas (sek.)	$\overline{\theta}$	Laikas (sek.)
tai040a	3139370	**0,052(3)**	**204,916**	0,059(2)	486,000
tai050a	4938796	0,192(**2**)	**268,705**	**0,131(2)**	2520,000
tai060a	7205962	0,215(1)	**713,455**	**0,144(2)**	4050,000
tai080a	13499184	**0,367**(0)	**3040,000**	0,426(0)	3948,000
tai100a	21043560	**0,311**(0)	6200,000	0,447(0)	**2646,000**

Pastaba. Laikas žymi vieno vykdymo vidutinį CP laiką. Skliausteliuose pateikiama, kiek kartų buvo rastas GŽS. Geriausia žinoma tikslo funkcijos reikšmė duomenų pavyzdžiui `tai100a` yra iš publikacijos (Misevičius, 2019).

14 lentelė. GHA ir memetinio algoritmo (MA) (Benlic, Hao, 2015) palyginimo rezultatai (III)

Duomenų pavyzdžiai	GŽR	GHA		MA	
		$\overline{\theta}$	Laikas (sek.)	$\overline{\theta}$	Laikas (sek.)
tai050b	458821517	**0,000(10)**	**5,488**	**0,000(10)**	72,000
tai060b	608215054	**0,000(10)**	**5,036**	**0,000(10)**	312,000
tai080b	818415043	**0,000(10)**	**29,161**	**0,000(10)**	1878,000
tai100b	1185996137	**0,000(10)**	**83,515**	**0,000(10)**	816,000
tai150b	498896643	**0,001(4)**	**2150,000**	0,060(1)	4686,000

Pastaba. Laikas žymi vieno vykdymo vidutinį CP laiką.

15 lentelė. GHA ir dažnų pėdsakų paieškos (DPP) algoritmo (Zhou ir kt., 2020) palyginimo rezultatai

Duomenų pavyzdžiai	GŽR	GHA		DPP	
		$\overline{\theta}$	Laikas (sek.)	$\overline{\theta}$	Laikas (sek.)
tai040a	3139370	0,052	**204,916**	**0,037**	3150,000
tai050a	4938796	0,192	**268,705**	**0,106**	4068,000
tai060a	7205962	0,215	**713,455**	**0,189**	3600,000
tai080a	13499184	**0,367**	**3040,000**	0,467	3112,000
tai100a	21043560	**0,311**	6200,000	0,390	**2166,000**
tai050b	458821517	**0,000**	**5,488**	**0,000**	12,000
tai060b	608215054	**0,000**	**5,036**	**0,000**	24,000
tai080b	818415043	**0,000**	**29,161**	**0,000**	84,000
tai100b	1185996137	**0,000**	**83,515**	**0,000**	174,000
tai150b	498896643	**0,001**	**2150,000**	0,092	2784,000

Pastabos. 1. Laikas žymi vieno vykdymo vidutinį CP laiką. 2. Originaliame (Zhou ir kt., 2020) straipsnyje laikas yra pateiktas minutėmis.

15 lentelė (tęsinys). GHA ir dažnų pėdsakų paieškos (DPP) algoritmo (Zhou ir kt., 2020) palyginimo rezultatai

Duomenų pavyzdžiai	GŽR	GHA		DPP	
		$\bar{\theta}$	Laikas (sek.)	$\bar{\theta}$	Laikas (sek.)
sko072	66256	**0,000**	**29,380**	**0,000**	288,000
sko081	90998	**0,000**	**95,421**	**0,000**	198,000
sko090	115534	**0,000**	229,456	0,010	**144,000**
sko100a	152002	**0,000**	542,640	**0,000**	**510,000**
sko100b	153890	**0,000**	**227,774**	**0,000**	348,000
sko100c	147862	**0,000**	**400,697**	**0,000**	522,000
sko100d	149576	**0,000**	**377,108**	**0,000**	972,000
sko100e	149150	**0,000**	**438,632**	**0,000**	732,000
sko100f	149036	**0,000**	790,550	0,003	**240,000**
wil100	273038	**0,000**	**600,566**	**0,000**	984,000
tho150	8133398	0,007	6300,000	**0,006**	**3444,000**

Pastabos. 1. Laikas žymi vieno vykdymo vidutinį CP laiką. 2. Originaliame (Zhou ir kt., 2020) straipsnyje laikas yra pateiktas minutėmis.

16 lentelė. GHA ir hibridinio genetinio algoritmo (HGA) (Drezner ir kt., 2005) palyginimo rezultatai (I)

Duomenų pavyzdžiai	GŽR	GHA		HGA	
		$\overline{\theta}$	Laikas (sek.)	$\overline{\theta}$	Laikas (sek.)
dre030	508	**0,000(10)**	**0,875**	**0,000**	143,400
dre042	764	**0,000(10)**	**9,809**	1,340	547,800
dre056	1086	**0,000(10)**	**86,024**	17,460	1810,800
dre072	1452	**0,000(10)**	**489,877**	27,280	5591,400
dre090	1838	**0,000(10)**	**4546,978**	33,880	11557,800
dre110	2264	**0,000(10)**	**16500,000**		
dre132	2744	**13,620(3)**	**30000,000**		

Pastaba. Laikas žymi vieno vykdymo vidutinį CP laiką. Skliausteliuose pateikiama, kiek kartų buvo rastas GŽS.

17 lentelė. GHA ir hibridinio genetinio algoritmo (HGA) (Drezner ir kt., 2005) palyginimo rezultatai (II)

Duomenų pavyzdžiai	GŽR	GHA		HGA	
		$\bar{\theta}$	Laikas (sek.)	$\bar{\theta}$	Laikas (sek.)
tai027e1	2558	**0,000**	**0,332**	**0,000**	~60,000
tai027e2	2850	**0,000**	**0,399**	**0,000**	~60,000
tai027e3	3258	**0,000**	**0,078**	**0,000**	~60,000
tai045e1	6412	**0,000**	**1,346**	**0,000**	~300,000
tai045e2	5734	**0,000**	**5,713**	**0,000**	~300,000
tai045e3	7438	**0,000**	**2,471**	**0,000**	~300,000
tai075e1	14488	**0,000**	**52,287**	**0,000**	~2220,000
tai075e2	14444	**0,000**	**25,134**	0,339	~2220,000
tai075e3	14154	**0,000**	**36,677**	**0,000**	~2220,000

Pastaba. Laikas žymi vieno vykdymo vidutinį CP laiką.

18 lentelė. GHA ir hibridinio genetinio algoritmo su diferencijuotu pagerinimu (HGA-DI) (Drezner, Misevičius, 2013) palyginimo rezultatai (I)

Duomenų pavyzdžiai	GŽR	GHA		HGA-DI	
		$\overline{\theta}$	Laikas (sek.)	$\overline{\theta}$	Laikas (sek.)
bl036	3296	**0,000(10)**	**9,491**	**0,000(10)**	51,000
bl049	4548	**0,229(3)**	230,746	0,334(0)	**125,000**
bl064	5988	**0,207(2)**	519,594	0,227(0)	**356,000**
bl081	7532	**0,319(1)**	1291,861	0,494(0)	**937,000**
bl100	9264	**0,479(1)**	**1963,669**	0,548(0)	2306,000
bl121	11400	**0,516(** 0)	22000,000	0,867(0)	**5510,000**
bl144	13460	**0,431(** 0)	43000,000	0,993(0)	**12490,000**

Pastaba. Laikas žymi vieno vykdymo vidutinį CP laiką. Skliausteliuose pateikiama, kiek kartų buvo rastas GŽS.

19 lentelė. GHA ir hibridinio genetinio algoritmo su diferencijuotu pagerinimu (HGA-DI) (Drezner, Misevičius, 2013) palyginimo rezultatai (II)

Duomenų pavyzdžiai	GŽR	GHA		HGA-DI	
		$\overline{\theta}$	Laikas (sek.)	$\overline{\theta}$	Laikas (sek.)
ci036	168611971	**0,000(10)**	**1,279**	**0,000(10)**	50,000
ci049	236355034	**0,000(10)**	**6,864**	**0,000(10)**	124,000
ci064	325671035	**0,000(10)**	**46,818**	**0,000(10)**	354,000
ci081	427447820	**0,000(10)**	**250,470**	**0,000(10)**	932,000
ci100	523146366	**0,000(10)**	9998,000	0,007(3)	**2285,000**
ci121	653409588	**0,026(2)**	52000,000	0,051(1)	**5431,000**
ci144	794811636	**0,050(1)**	80000,000	0,068(**1**)	**12430,000**

Pastaba. Laikas žymi vieno vykdymo vidutinį CP laiką. Skliausteliuose pateikiama, kiek kartų buvo rastas GŽS.

4.5 Apibendrinamosios pastabos

Šiame skyriuje iš pradžių pateikiami KP uždavinio testinių duomenų pavyzdžiai, su kuriais ir atlikti kompiuteriniai eksperimentai. Pristatomi genetinio-hierarchinio algoritmo efektyvumo įvertinimo ir palyginimo kriterijai, taip pat pagrindiniai GH algoritmo naudoti parametrai.

Tuomet nuosekliai aprašomi eksperimentai su kryžminimo procedūromis bei mutavimo procedūromis; pateikiami gauti rezultatai. Buvo atlikti ir išplėstiniai eksperimentai su įvairiais papildomais GH algoritmo konfigūraciniais variantais bei parametrų reikšmėmis. Pateikiamos išsamios gautų rezultatų lentelės su atitinkamomis pastabomis ir išvadomis.

Pagrindinis pastebėjimas yra tas, kad GH algoritmo gaunamų sprendinių kokybė pastebimai pagerėja, suformavus tinkamą algoritmo komponentų ir parametrų konfigūraciją. Net 88 tirti testiniai duomenų pavyzdžiais yra išspręsti (pseudo-)optimaliai, o vidutinis (pseudo-)optimalių sprendinių suradimo laikas neviršija 15 s.

Skyriaus pabaigoje pateikti GH algoritmo palyginimo su kitais algoritmais rezultatai, kurie paliudija GH algoritmo konkurencingumą, o ir pranašumą kitų algoritmų atžvilgiu reikšmingai daliai tirtų testinių duomenų pavyzdžių.

5 IŠVADOS

1. Nustatyta, kad iteracinių euristinių algoritmų taikymas, sprendžiant sunkius kombinatorinio optimizavimo uždavinius, yra progresyvus būdas, siekiant surasti aukštos kokybės sprendinius per priimtiną skaičiavimų laiką. Po atliktos KP uždaviniui taikomų algoritmų apžvalgos tyrimo objektu buvo pasirinkti kelių lygmenų iteratyvieji, t. y. hierarchiniai euristiniai algoritmai.
2. Pasiūlytas ir realizuotas inovatyvus genetinis hierarchinis algoritmas (GHA) KP uždaviniui. Esminės GHA savybės yra tokios: algoritme panaudotos įvairaus efektyvumo sprendinių kryžminimo procedūros; algoritme integruota kelių lygmenų hierarchinė iteratyvioji tabu paieška; hierarchinės iteratyviosios tabu paieškos algoritme įkomponuotos originalios sprendinių mutavimo procedūros, užtikrinančios paieškos proceso diversifikavimą; populiacijos atnaujinimas vykdomas, išlaikant pakankamą populiacijos sprendinių įvairovės laipsnį; inkorporuota populiacijos perkrovimo procedūra leidžia sumažinti genetinės paieškos stagnacijos neigiamas pasekmes.
3. Gauti kompiuterinių eksperimentų rezultatai patvirtina GH algoritme integruoto hierarchinio iteratyviosios tabu paieškos algoritmo su daugeliu hierarchinių lygmenų prasmingumą.
4. Atlikti GH algoritmo efektyvumo eksperimentiniai tyrimai leidžia daryti esminę išvadą, jog sudarytas algoritmas yra tinkamas, našus būdas spręsti KP uždavinį. Eksperimentuose tirtiems 88 testiniams duomenų pavyzdžiams GH algoritmo vidutinis (pseudo-)optimalių sprendinių suradimo laikas yra lygus 14,87 s.
5. Gauti algoritmų palyginimo eksperimentų rezultatai liudija GH algoritmo pranašumą, lyginant su kitais šiuolaikiniais, moderniais euristiniais algoritmais. Intensyviai tirtam testinių duomenų rinkiniui, sudarytam iš 20 duomenų pavyzdžių, vidutinis gautų tikslo funkcijos reikšmių nuokrypis yra lygus 1,59 %. Vidutinis algoritmo vykdymo laikas lygus ~60 s. Vidutinis tikslo funkcijos reikšmių nuokrypis yra mažesnis negu 0,5 % daugiau negu 70 % visų tirtų atvejų.
6. Paminėtinos kai kurios tolesnių tyrimų kryptys, tarp jų: tabu paieškos ir hierarchinės iteratyviosios tabu paieškos iteracijų skaičiaus sukalibravimas, galimai „intelektualesnių“ priemonių panaudojimas populiacijos perkrovimo procedūroje; populiacijos perkrovimų stiprumo, intensyvumo ir perkrovimų vykdymo dažnio subalansavimas.

6 SUMMARY

INTRODUCTION

The solving of hard optimization problems still poses real challenges for the researchers. Among such problems, the quadratic assignment problem (QAP) attracts the attention of many scientists. There are no efficient exact solution methods for the medium- and large-sized QAP test data instances; thus, the heuristic algorithms are widely applied.

The heuristic algorithms do not guarantee finding globally optimal solutions, but they enable to achieve high quality or near-optimal solutions within reasonable computation times.

One of the most promising groups of heuristic algorithms is the so-called hierarchical (hierarchicity-based) heuristic algorithms (HHA) (or simply hierarchical algorithms (HA)). The main principle of functioning of hierarchical heuristic algorithms is the multiple reuse of well-known efficient algorithms, like local search or tabu search algorithms. The main distinguishing feature of the hierarchical algorithms is the hierarchicity-based structure (architecture) of these algorithms. There are several hierarchical levels in the hierarchical algorithms. Usually, the current level of the hierarchical algorithm consists of the following three essential constituent parts (components, ingredients): i) selection of candidate solution, i.e., candidate acceptance; ii) local improvement, i.e., optimization of the current selected candidate solution; iii) perturbation of the obtained (improved) solution. Despite the fact that the structure of hierarchical algorithm is quite simple, this kind of algorithms distinguishes with quite high quality of the found (produced) solutions. This sort of heuristic algorithms, in particular, are investigated in this dissertation.

The object of the research. The object of the research is the hierarchical heuristic algorithms (in particular, genetic hierarchical algorithms) for the quadratic assignment problem.

The aim of the research. The aim of the research is to computationally investigate the performance of genetic hierarchical algorithms and their modifications for solving the quadratic assignment problem. Moreover, one wishes to find (determine) the most effective configurations (i.e., combinations of algorithmic ingredients and values of the controlling parameters) for the genetic hierarchical algorithms.

The tasks of the research. In order to achieve the aim of the research, the following tasks (objectives) have been set:

1. Exploratory studying of the state-of-the-art heuristic methods that are oriented towards combinatorial optimization and, in particular, the quadratic assignment problem.
2. Designing and implementation (including program testing, validation) of particular hierarchical heuristic algorithms for the QAP.
3. Performing extensive computational experiments with the created (implemented) heuristic algorithms for the QAP and their variants

(improvements). Analysing the speed of the algorithms and the quality of obtained results (solutions).

4. Determining the most promising variants and modifications of the proposed heuristic algorithms with respect to the run times of algorithms and the quality of the results that have been found during the experimentation.

The methods of the research. The research methodology is based on the theory (foundations) of combinatorial optimization, the adoption of heuristic optimization algorithms, algorithm analysis, numerical analysis.

Scientific novelty and practical relevance. The scientific novelty is due to three main aspects (facts).

1. In this work, there is proposed a new, hierarchicity-based heuristic algorithm, in particular, the genetic hierarchical algorithm (GHA) for the solution of the quadratic assignment problem.
2. Also, there is created a hierarchical iterated tabu search (HITS) algorithm, based on several hierarchical levels (parts), which can be used autonomically and be included (integrated) into the hybrid genetic algorithm (HGA). Thus, the HITS procedure can serve as an excellent local optimizer for the solutions produced by the crossover operator of the genetic algorithm.
3. In addition, an original novel genetic crossover operator (called a universal crossover) is proposed.

The created hierarchical genetic algorithm is implemented in the C# programming language as an autonomic executable module capable of dealing (solving) with the real QAP data instances of size up to 150.

Approbation of the research results. The main results of the dissertation were published in 4 scientific publications: 2 in the journals included in the list of scientific international databases (indexed in the Web of Science with Impact Factor). The results were as well presented at 3 scientific conferences.

The structure and volume of the dissertation. The dissertation consists of an introduction, 4 main chapters, conclusions, a list of references, a list of the author's publications. The total volume of the dissertation is 124 pages, including 22 figures, 19 tables and 198 references.

RELATED WORK

For solving the quadratic assignment problem, many various optimization techniques and approaches can be applied, including the exact, approximation and heuristic algorithms (Çela, 1998). The exact algorithms are only applicable to the problems (problem instances) of small sizes (Hahn et al., 2012).. For larger problems, heuristic algorithms are usually employed. A very huge spectrum of general-purpose heuristic methods can be potentially adopted:

i) local search (LS) algorithms (Aarts, Lenstra, 1997);
ii) simulated annealing (SA) (Kirkpatrick et al., 1983);
iii) tabu search (TS) (Glover, Laguna, 1997);
iv) genetic algorithms (GA) (Goldberg, 1989);
v) iterated (local) search (IS, ILS) (Lourenco et al., 2002);

vi) hierarchical (iterated local) search (HS, HILS) (Hussin, Stützle, 2009).

Going into more details, the LS algorithms (Aarts, Lenstra, 1997) are relatively quite simple, but effective enough in some cases; they may also be thought of as the origin of the more intelligent optimization techniques. There are two main types of the LS algorithms, i.e., a) a "greedy descent", and b) a "steepest descent". In both cases, the decision – to replace the current solution by the new one (i.e., to move to the neighbouring solution), or not – is positive if only new solution is better than the current one (i.e., the difference in the objective function values, Δz, is negative ($\Delta z = z(s') - z(s) < 0$, where $s' \in N(s)$, z is an objective function, $N(s)$ denotes a "neighbourhood" of the current solution, s)). It is assumed that the goal is to minimize the objective function. The difference between these LS groups is as follows: in the greedy descent, a move is performed to the first better solution from the neighbourhood of the current solution; in the steepest descent, the current solution is replaced by the best improving neighbouring solution. In both cases, the search process is continued until the current solution, s, is locally optimal.

Simulated annealing (Kirkpatrick et al., 1983) originates in statistical mechanics. It is based on the Monte Carlo model that is used to simulate the energy levels in cooling solids. Boltzmann's law is used to determine the probability of accepting a perturbation, resulting in a change ΔE in energy at the current temperature t, i.e., $P = \begin{cases} 1, & \Delta E < 0 \\ e^{-\Delta E / C_B t}, & \Delta E \geq 0 \end{cases}$, where C_B is a Boltzmann's constant. Starting from 1982, many authors applied simulated annealing to solve combinatorial optimization problems. The principle of SA algorithm is simple: start from a random solution. Given a solution s, select a neighbouring solution s' and compute the difference in the objective function values, $\Delta z = z(s') - z(s)$. If the objective function value is improved ($\Delta z < 0$), then replace the current solution by the new one. If $\Delta z \geq 0$, then accept a move with probability $P(\Delta z) = e^{\frac{-\Delta z}{t}}$, where t is the current temperature (Boltzmann's constant is not required when applying the algorithm to combinatorial problems). Regarding the above probabilistic acceptance, it is achieved by generating a random number in $[0, 1]$ and comparing it against the threshold $e^{\frac{-\Delta z}{t}}$ (where the exponential function plays the role of acceptance function). The procedure is repeated until a stopping condition is satisfied, for example, a predefined number of trials have been performed. As a resulting solution, usually, the "best so far" solution is returned by the algorithm.

Tabu search meta-heuristic (Glover, Laguna, 1997) was introduced by Hansen, Jaumard (1987) and Glover (1989). Since then, TS has been proven to be among the most powerful intelligent techniques for difficult optimization problems. Briefly speaking, TS is based on the neighbourhood search with local-optima avoidance in a rather deterministic way. The key idea of TS is to allow climbing moves when no improving move exists. However, some moves have to be forbidden in order to avoid cycling of the search procedure. The tabu search process starts from the initial solution s, maybe, randomly generated in a search space, $S = \{s, s', s'', s''', \ldots\}$, and moves repeatedly from the current solution to the neighbouring one. At each step of the

process, a set (subset) $N'(s)$ of the neighbouring solutions of the current solution s is considered, and the move that improves most the objective function value z is chosen. If there are no improving moves, the tabu search chooses the one that least degrades the objective function, i.e., a move is performed to the best neighbour s' in $N'(s)$ (even if $z(s') > z(s)$). For more details, see sub-section 3.4.

The original concepts of genetic algorithms (Goldberg, 1989), which are based on the biological process of natural selection, were developed in 1975 by Holland. The power of GAs has been demonstrated for many optimization problems. GA operates with some group (P) (called a population) of solutions (called individuals) from $S = \{s, s', s'', s''', ...\}$. (This is quite different from the above three approaches, which can be viewed as single solution-based heuristics.) Each individual (s), corresponding to the solution of the optimization problem, is associated with some fitness, corresponding to the objective function value ($z(s)$). In case of minimization problem, the less is the objective function value, the more fit is the individual, and the larger is the probability that the individual will survive in the evolution process. Over many generations, the best fitting individuals tend to predominate, while less fit individuals tend to die. GAs can be characterized by the following features: a) a mechanism for selecting individuals from the population; b) an operator for creating new individuals by combining the information contained in the previous individuals; c) a procedure for generating new solution by random perturbations of the single solution; d) a rule for updating the population. These features are referred to as selection, crossover, mutation and replacement, respectively. For more details, see Chapter 3.

The key idea of IS/ILS (Lourenco et al., 2002) is to obtain better results by a reconstruction (destruction) of the existing solution and the following improvement (rebuilding) procedure. By applying this type of process in an iterative way, one seeks for higher quality solutions. In the first phase of the process, one reconstructs (mutates) the existing solution; the rationale is that continuing search from the reconstructed solution may allow to escape from a local optimum and try to find better solutions. In the second phase, one tries to improve the solution just ruined as best as one can; hopefully, the new solution is better than the solution obtained in the previous phase. Also, see sub-section 3.4.

Regarding the hierarchical heuristic algorithms, the details are described in sub-section 3.4.

More precisely, the following heuristic algorithms for quadratic assignment problem were proposed in the literature: local search based algorithm (Angel, Zissimopoulos, 1998), simulated annealing algorithm (Paul, 2011), tabu search methodology-based algorithm (Taillard, 1991), genetic algorithms (including the hybrid genetic algorithms) (Misevicius, 2004; Benlic, Hao, 2015), iterated (breakout) local search (Benlic, Hao, 2013), iterated tabu search (Misevicius, 2012), greedy randomized adaptive search procedure (GRASP) (Li et al., 1994), frequent pattern-based search (Zhou et al., 2020), other heuristic algorithms (among them, swarm intelligence based and nature-inspired algorithms: ant colony optimization algorithm (Gambardella et al., 1999), artificial bee colony optimization algorithm (Dokeroglu at al.,

2019), particle swarm optimization algorithm (Hafiz, Abdennour,2016), teaching-learning based optimization algorithm (Dokeroglu, 2015), biogeography-based optimization algorithm (Lim at al., 2016)).

So far, the most promising results were achieved by applying the iterated local search/tabu search-based algorithms (Bencil, Hao, 2013; Misevicius, 2012) and hybrid (combined) algorithms, which, in particular, combine the genetic algorithms operations (crossover, mutation) and local search/tabu search-based procedures (Misevicius, 2004; Bencil, Hao, 2015).

GENETIC HIERARCHICAL ALGORITHM FOR THE QUADRATIC ASSIGNMENT PROBLEM

The quadratic assignment problem

The quadratic assignment problem can be stated as follows (Çela, 1998; Koopmans, Backmann, 1957). Given two integer matrices, $\boldsymbol{A} = (a_{ij})_{n\times n}$, $\boldsymbol{B} = (b_{kl})_{n\times n}$, and the set Π of all possible permutations of the integers from 1 to n, find a permutation $p = (p(1), p(2), \dots, p(n)) \in \Pi$ that minimizes the following objective function:

$$z(p) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} b_{p(i)p(j)}. \quad (1)$$

In the context of the location theory, the QAP formulation is used to model the problem of allocation n facilities (objects) to n locations (sites) with the objective to minimize the cost associated with the distance and flow between the facilities (Koopmans, Backmann, 1957).The matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, in particular, correspond to the flow and distance matrices, respectively.

Some preliminary (basic) formal definitions are as follows.

1. Suppose that $p(u)$ $(u = 1, \dots, n)$ and $p(v)$ $(v = 1, \dots, n,\ u \neq v)$ are two elements of the permutation p. Then, $p^{u,v}$ is defined in the following way:

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

This means that the permutation $p^{u,v}$ is obtained from the permutation p by interchanging exactly the elements $p(u)$ and $p(v)$ in the permutation p. It should be noted that the following equations hold: $p^{u,u} = p$, $p^{u,v} = p^{v,u}$, $(p^{u,v})^{u,v} = p$.

2. The difference in the objective function (z) values when the permutation elements $p(u)$ and $p(v)$ are interchanged is calculated according to the following formula:

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

The difference in the objective function values for the permutations p and p' ($p' = p^{u,v}$) can be calculated faster, under condition that the previously calculated differences ($\Delta(p^{i,j}, p)$ $(i, j = 1, \dots, n)$) are memorized (stored in a matrix Δ). The difference value is calculated by using $O(1)$ operations (Taillard, 1991; Frieze et al., 1989). After the interchange of the elements $p(u)$ and $p(v)$ has been performed, new differences $\Delta'(p^{i,j}, p)$ ($i \neq u$, $i \neq v$, $j \neq u$, $j \neq v$) are calculated according to this equation:

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

Note. If $i = u$ or $i = v$ or $j = u$ or $j = v$, then the formula (3) should be applied. Thus, all differences are calculated using only $O(n^2)$ operations. Still, the very initial calculation of the values of the matrix Δ requires $O(n^3)$ operations (but only once before starting the optimization algorithm).

If the matrix $\boldsymbol{A}$ and/or matrix $\boldsymbol{B}$ are symmetric, then the formula (3) becomes simpler. If it is assumed that the matrix $\boldsymbol{B}$ is symmetric. Then, the (asymmetric) matrix $\boldsymbol{A}$ can be transformed to the symmetric matrix $\boldsymbol{A'}$. Thus, the following formula is obtained:

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

where, $a'_{uk} = a_{uk} + a_{ku}$, $u = 1, \dots, n$, $v = 1, \dots, n$, $u \neq v$. Analogously, the formula (4) turns to the following formula:

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

If $i = u(v)$ or $j = u(v)$, the formula (5) must be applied.

If it is supposed that one dispose of 3-dimensional matrices $\boldsymbol{A''} = (a''_{uvk})_{n\times n\times n}$ and $\boldsymbol{B''} = (b''_{lrt})_{n\times n\times n}$; moreover, let $a''_{uvk} = a'_{uk} - a'_{vk}$, $b''_{lrt} = b_{lt} - b_{rt}$, $l = p(j)$, $r = p(i)$, $t = p(k)$. Then, the following formulas can be applied for the calculation of differences in the objective function values:

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

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

Using the matrices $\boldsymbol{A''}$ and $\boldsymbol{B''}$ (instead of $\boldsymbol{A'}$ and $\boldsymbol{B}$) allows to save up to 20% of computation (CPU) time.

3. The distance (Hamming distance) between two permutations p_1 and p_2 is defined as $\delta(p_1, p_2) = |\{i: p_1(i) \neq p_2(i)\}|$. The following equations hold: $\delta(p, p) =$

$0,\ \delta(p_1,p_2)\neq 1,\ 0\leq\delta(p_1,p_2)\leq n,\ \delta(p_1,p_2)=\delta(p_2,p_1),\ \delta(p,p^{u,v})=2$. For k different $u_1,u_2,\dots,u_k$, this equation holds: $\delta\big((((p^{u_1,u_2})^{u_2,u_3})^{\cdots})^{u_{k-1},u_k}\big)=\mu$.

4. The neighbourhood of solution p (pairwise interchange neighbourhood) Θ_2 is determined by using the formula: $\Theta_2(p)=\{p':p'\in\Pi_n,p'\neq p,\delta(p,p')\leq 2\}$. That is, the neighbourhood $\Theta_2(p)$ consists of the solutions: $p^{1,2}$, $p^{1,3}$, …, $p^{1,n}$, $p^{2,3}$, …, $p^{2,n}$, …, $p^{i-1,n}$, $p^{i,i+1}$, …, $p^{i,n}$, …, $p^{n-1,n}$. Thus, the size of the neighbourhood Θ_2 ($|\Theta_2|$) is equal to $\big(n(n-1)\big)/2$. Thus, the exploring (scanning) of the neighbourhood Θ_2 requires $O(n^2)$ operations. The set of all possible neighbourhoods can be seen as a solution space (search space).

5. The solution $p^{\bullet}\in\Pi$ is said to be locally optimal with respect neighbourhood Θ if for every solution p' from the neighbourhood $\Theta(p^{\bullet})$, the following inequality holds: $z(p^{\bullet})\leq z(p')$.

6. The crossover operator can be formally described by using the operator ψ: $\Pi\times\Pi\to\Pi$, such as:

$$p^{\circ}=\psi(p',p'')\neq p'\vee p^{\circ}=\psi(p',p'')\neq p'',\text{ if } p'\neq p''; \tag{9}$$

where p', p'', p° are parental solutions, p° denotes the offspring solution. (In principle, the offspring may be identical to one of the parents if the parents are very similar.) The permutation can be associated with the individual's chromosome; then, the permutation element $p(i)$ would correspond to a gene, occupying the *i*th locus of the chromosome. The crossover operator must ensure that the offspring's chromosome inherits the genes, which are the same for both parents, i.e.:

$$p'(i)=p''(i)\Rightarrow p^{\circ}(i)=p'(i)=p''(i),i=1,2,\dots,n; \tag{10}$$

where, again, p', p'', p° are parental solutions and the offspring solution, respectively.

7. The mutation process can be defined by the use of the formal operator ϕ: $\Pi\to\Pi$. Thus, if $p^{\sim}=\phi(p)$, then $p^{\sim}\in\Pi$; be to, $p^{\sim}\neq p$. Even more formalized operator is as follows: $\phi(p,\mu)$: $\Pi\times\mathrm{N}\to\Pi$, which transforms the current solution p to the new solution $p^{\sim}$, such as $\delta(p,p^{\sim})=\delta(p,\phi(p))=\mu$. The number (parameter) μ is referred to as a mutation rate. Obviously, $2\leq\mu\leq n$. Thus, the mutation process with the rate μ affects $100\mu/n$ per cent elements of the solution (permutation). A move (μ-move) is said to be performed. As a special case, the 2-move is defined as follows: $\phi(p,u,v)$: $\Pi\times\mathrm{N}\times\mathrm{N}\to\Pi$, which swaps the *u*th and *v*th element of the permutation, i.e., $\phi(p,u,v)=p^{u,v}$.

Genetic hierarchical algorithm for the quadratic assignment problem

The following are the main parts (components) of the proposed algorithm (GHA):

– (initial) population construction;

– selection of parental solutions for the crossover procedure;

– crossover procedure (operator);

– improvement of the produced solution by using the hierarchical iterated tabu search (HITS);

– mutation procedure (integrated with the HITS algorithm);

– population replacement (update);

– population restart (renovation).

```
procedure Genetic_Hierarchical_Agorithm;
//input: n, A, B
//output: p* – the best found solution (permutation)
//parameters: PS – population size, N_gen – number of generations,
//            DT – distance threshold, L_idle_gen – idle generations limit
begin
  get data matrices, parameters;
  initialize algorithm variables;
  generate sorted INITIAL POPULATION P of size PS;
  p* := argmin_{p∈P} {z(p)};  //memorize the best solution of the initial population
  gen_index := 1;
  while gen_index <= N_gen do begin
    perform SELECTION procedure to choose the parents p', p'' ∈ P;
    produce offspring by applying CROSSOVER procedure to p', p'';
    apply HIERARCHICAL ITERATED TABU SEARCH to the produced
    offspring p°, get an (improved) solution p☆;
    if z(p☆) < z(p*) then p* := p☆;  //memorize the best found solution
    add p* to a pool of offspring;
    if number of idle generations exceeds the predefined limit
       L_idle_gen then begin
       POPULATION RESTART;  //perform population restart procedure
       if min_{p∈P} {z(p)} < z(p*) then p* := argmin_{p∈P} {z(p)}
    end  //of if
    else perform POPULATION REPLACEMENT;  //update and sort the current population
    gen_index := gen_index +1  //go to the next generation
  endwhile
end.
```

Note. There also are several control parameters for the local optimizer (the hierarchical iterated tabu search algorithm).

Figure 1. Pseudocode of the genetic hierarchical algorithm

The created algorithm distinguishes with these essential features.

I. For the improvement of the produced offspring by the crossover procedure, the hierarchical iterated tabu search (HITS) algorithm is applied. The HITS algorithm is in turn based on the use of several levels of tabu search algorithm. The maximum number of levels is equal to 7.

II. Within the HITS algorithm, the tabu search process (search intensification) is effectively combined with the solution perturbation, i.e., mutation process (search diversification).

III. The genetic algorithm operates with both high quality and diversified population of solutions. In order to obtain the (initial) population as best as possible, the HITS algorithm with the substantially increased number of iterations is applied during the phase of construction of the starting (initial) population.

IV. For the replacement of population, the so-called minimum distance criterion is applied.

V. In order to eliminate the possible stagnation of the search (genetic) process, the population restart procedure is incorporated into the genetic algorithm. The restart procedure is activated only in the cases of identification of stagnated process (for example, the population does not change over some interval of time).

The pseudocode of the genetic hierarchical algorithm is presented in Fig. 1.

The crossover procedures used in the genetic algorithm

The following crossover procedures within the genetic algorithm were investigated:

- one-point crossover — 1PX;
- two-point crossover — 2PX;
- uniform crossover — UX;
- shuffle crossover — SX;
- partially-mapped crossover — PMX;
- swap-path crossover — SPX;
- heuristic swap-path crossover — HSPX;
- swap-path descent crossover — SPDX;
- cycle crossover — CX;
- cohesive crossover — COHX;
- universal crossover — UNIVX;
- multi-parent crossover — MPX.

Hierarchical iterated tabu search algorithm

In the case of the tabu search (TS), first, the iterated tabu search—ITS—is obtained by combining the tabu search and some perturbations. Further, the ITS algorithm itself is combined with another ITS algorithm, which results in the "ITS-ITS" algorithm.

The essential property of hierarchical iterated tabu search algorithm is the hierarchicity-like appearance (form) of the algorithm's structure. This feature is also known as self-similarity. To sum up, the principle of the HITS algorithm is that the higher-level (for example, *k*th) algorithm uses the lower-level (for example, *k*–1-th level) algorithm, *k*–1-th level uses *k*–2-level, etc. (all the way down to the 0th level, which, in fact, is the self-contained tabu search procedure) (see Fig. 2). Each level contains three main parts. As an example, *k*th-level contains: i) *k*–1-th level hierarchical iterated tabu search algorithm; ii) candidate acceptation; iii) perturbation procedure applied to the accepted candidate solution. (Usually, the last found best solution is accepted as a candidate for mutation.)

The pseudocode of the *k*-level hierarchical iterated tabu search algorithm is presented in Fig. 3.

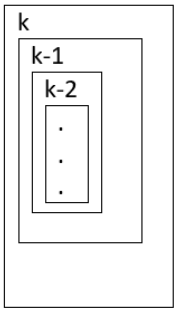


Figure 2. Self-similar structure of the hierarchical algorithm

In the case of 1-level HITS algorithm (iterated tabu search algorithm), there are two levels: 1st-level algorithm directly uses the 0-level algorithm, i.e., the self-contained tabu search procedure. The pseudocode of the iterated tabu search algorithm is shown in Fig. 4.

```
procedure k-Level_Hierarchical_Iterated_Tabu_Search;
//input: p° – initial solution (permutation)
//output: p^(k) – the best found solution
//control parameters: Q^(k) – number of iterations; μ – mutation rate
begin
  calculate the differences in the objective function values
  Δ((p°)^{i,j}, p°), i, j = 1, ..., n;
  initialize tabu list T;
  p := p°;  p^(k) := p°;
  for q^(k) := 1 to Q^(k) do begin
    apply procedure k–1-Level_Hierarchical_Iterated_Tabu_Search
    to the solution p, get (improved) solution p^◊;
    if z(p^◊) < z(p^(k))  p^(k) := p^◊;  //the best found solution is memorized
    if q^(k) < Q^(k) then begin
       p := Candidate_Acceptance(p^◊, p^(k));
       apply mutation procedure to the solution p, with the
       mutation rate μ, return the mutated solution p~;
       p := p~
    endif
  endfor
end.
```

Figure 3. Pseudocode of the *k*-level hierarchical iterated tabu search algorithm

```
procedure Iterated_Tabu_Search;
//input: p – current solution (permutation)
//output: p^∇ – the best found solution (along with differences of the objective function (Δ))
//control parameters: Q – number of iterations, μ – mutation rate
begin
  p^∇ := p;
  for q := 1 to Q do begin
    apply procedure Tabu_Search to the solution p,
    get (improved) solution p•;
    if z(p•) < z(p^∇) p^∇ := p•; //the best found solution and the differences (Δ) are memorized
    if q1 < Q1 then begin
       p := Candidate_Acceptance(p•, p^∇);
       apply mutation procedure to the solution p, with the
       mutation rate μ, return mutated solution p~;
       p := p~
    endif
  endfor
end.
```

Figure 4. Pseudocode of the iterated tabu search algorithm

```
procedure Tabu_Search;
//input: p – current solution (permutation) (with differences of the objective function (Δ)),
//        n – problem size
//output: p• – the best found solution (with differences of the objective function (Δ))
//control parameters: τ – number of iterations of tabu search, h – tabu tenure,
//        α – randomization factor, β – idle iteration limit factor
begin
  p• := p; k := 1; k' := 1; archive_counter := 0; L := ⌊βτ⌋;
  while (k ≤ τ) begin //main cycle of the tabu search procedure
    delta'min := ∞; delta''min := ∞; u' := 1; v' := 2;
    //scanning the neighbourhood Θ2(p), finding the best not forbidden solution
    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);
        aspiration := iif(z(p) + delta < z(p•), TRUE, FALSE);
        if ((delta < delta'min) and (tabu = FALSE)) or (aspiration = TRUE)
        then begin
          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 //the „second" solution is included into the archive Γ
      archive_counter := archive_counter + 1; Γ[archive_counter] ← p, Δ, u'', v''
    endif;
    if delta'min < ∞ then begin
      p := p^{u',v'}; //the current solution is replaced by the new one
      recalculate the differences of the objective function
      Δ(p^{i,j}, p), i, j = 1, ..., n;
      if z(p) < z(p•) then begin
        p• := p; k' := k endif; //the best found solution is memorized
      t_{u',v'} := k + h //the interchange of elements p(u'), p(v') is forbidden for h future iterations
    endif;
    if (k − k' > L) and (k < τ − L) then begin
      //restart tabu search process using a solution from the archive
      clear tabu list T;
      random_access_index :=
      random(archive_counter * 0.8, archive_couter);
      p, Δ, u'', v'' ← Γ[ random_access_index];
      p := p^{u'',v''};
      recalculate the differences of the objective function
      Δ(p^{i,j}, p), i, j = 1, ..., n;
      t_{u'',v''} := k + h; k' := k
    endif;
    k := k + 1
  endwhile;
  clear tabu list T
end.
```

Notes. 1. Immediate "**if**" function **iif**(x, y_1, y_2) returns y_1, if x = TRUE; otherwise, it returns y_2.
2. Function **random**() generates (pseudo-)random real number within the interval [0, 1].
3. Function **random**(x_1, x_2) generates (pseudo-)random integer between x_1 and x_2.

Figure 5. Pseudocode of self-contained tabu search procedure

The self-contained tabu search procedure plays a central role in the hierarchical ITS algorithm and can be thought of as a "kernel" (or "engine") of HITS. The TS procedure explores in an iterative way the neighbourhood of the current solution and chooses a move that improves most the objective function value. The moves that are not forbidden are only taken into consideration. The tabu list is then updated, and the current solution is replaced by the new one, which is used as a starting point for the next iteration. More precisely, TS is based on the use of the neighbourhood Θ_2 as well as the tabu list T, which is organized as a two-dimensional matrix (matrices $T = (t_{ij})_{n\times n}$). The entry t_{ij} stores the current iteration number plus the tabu tenure (prohibition interval), h, i.e., the number of the future iteration, starting at which the corresponding elements of the permutation may again be interchanged (we used $h = 0.3n$). (If the entry entry t_{ij} is equal or greater than the current iteration number, then the corresponding move is considered as forbidden.) However, the tabu status is ignored if the aspiration criterion is met, i.e., the move results in a solution that is better than the best so far solution.

In addition to the tabu list, the additional (alternative) memory like mechanism is used as well. This is to maintain an archive of good solutions that were evaluated but not chosen. The goal is to diversify the search process and explore more regions of the search space. In order to implement this mechanism, an archive Γ is used. In particular, the archive Γ contains the so-called "second" solutions (not the overall best solutions, but the second-best solutions found during the current iteration after scanning the neighbourhood Θ_2). Then, if the best so far found solution does not change more than $\lfloor\beta\tau\rfloor$ iteration in a row, the tabu list is emptied, and the search is restarted from one of the second solutions from Γ (where, τ is the number of tabu search iteration, and β denotes the idle iteration limit parameter such as $1 \le \lfloor\beta\tau\rfloor \le \tau$). The complexity of the TS algorithm is proportional to $O(n^2)$. The pseudocode of the self-contained tabu search procedure is presented in Fig. 5.

The used mutation procedures

The following mutation procedures (integrated within the hierarchical tabu search algorithm instead of the genetic algorithm) were considered:

- random pairwise interchange based mutation — M1;
- random key mutation — M2;
- opposition-based mutation — M3;
- distance-based mutation — M4;
- real-discrete (smallest positive value) mutation (I) — M5;
- modified real-discrete (smallest positive value) mutation (II) — M6;
- combined mutation (combination of M3 and M6) — M7;
- greedy search based mutation — M8;

- greedy randomized search based mutation — M9;
- randomized local search based mutation — M10;
- randomized partial local search based mutation — M11;
- explorative tabu search using "second" solution — M12.

COMPUTATIONAL EXPERIMENTS

The computational experiments were performed by using the genetic algorithm, which was implemented in C# programming language. The experiments were carried out on a 3 GHz personal computer.

In the computational experiments, we have used the test (benchmark) data (instances) from the electronic public library of the QAP instances—QAPLIB (Burkard et al., 1989).

In the experiments, the following criterion for the evaluation of performance of the genetic hierarchical algorithm was used, i.e., the average relative percentage deviation ($\bar{\theta}$) of the found solution from the best known (pseudo-optimal) solution (BKS) (BKSs are from QAPLIB).

The average relative percentage deviation $\bar{\theta}$ is calculated in accordance with the following formula: $\bar{\theta} = 100(\bar{z} - z_{BKV})/z_{BKV}[\%]$; where $\bar{z}$ is the average value of the found values of the objective function averaged over W independent runs of the genetic hierarchical algorithm (we used $W = 10$); z_{BKV} denotes the best known (pseudo-optimal) value (BKV) of the objective function (BKVs are from QAPLIB).

Three groups (setups) of the experiments were carried out: i) comparative experiments with different crossover procedures; ii) comparison of distinct mutation procedures; iii) additional extended experiments with different values of control parameters of the GH algorithm.

In the first case, the number of generations of GHA (N_{gen}) is equal to 100, and the population size (PS) is equal to 10. The number of tabu search iterations (τ) is equal to 20, the number of iterations of the hierarchical tabu search procedure is equal to 2. The number of the hierarchical levels is equal to 7. The genetic algorithm was run 10 times on every benchmark instance.

The results of the experiments are presented in Table 1.

Table 1. Results of the comparison of crossover procedures

	Average relative deviation											
Cross. var.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bl049	0.730	0.765	0.712	0.756	0.756	0.812	0.792	0.73	0.765	0.712	0.668	0.853
bl064	1.009	0.775	0.768	1.222	1.062	0.795	1.136	1.096	0.882	0.962	0.962	1.416
ci049	0.004	0.002	0.009	0.000	0.012	0.004	0.000	0.000	0.003	0.013	0.005	0.003
ci064	0.092	0.086	0.103	0.087	0.081	0.097	0.066	0.055	0.085	0.068	0.098	0.11
dre042	8.770	15.052	13.665	11.99	11.126	18.77	9.738	8.586	14.058	14.607	7.408	7.696
dre056	28.785	28.674	36.206	37.661	35.47	39.282	38.122	38.471	29.963	24.199	26.298	46.335
lipa070a	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165
lipa070b	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
sko056	0.002	0.001	0.018	0.000	0.017	0.021	0.002	0.000	0.002	0.000	0.001	0.014
sko064	0.006	0.000	0.022	0.000	0.000	0.016	0.000	0.000	0.000	0.000	0.006	0.015
tai035a	0.355	0.416	0.332	0.263	0.233	0.506	0.313	0.239	0.386	0.252	0.484	0.215
tai035b	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
tai040a	0.483	0.417	0.556	0.477	0.482	0.686	0.464	0.462	0.513	0.622	0.601	0.534
tai040b	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
tai045e1	3.253	1.366	2.944	1.307	1.285	2.682	0.172	1.597	3.506	2.327	1.482	6.323
tai050a	0.810	0.834	0.872	0.742	0.712	0.96	0.784	0.884	0.829	0.797	0.912	0.838
tai050b	0.000	0.033	0.033	0.000	0.000	0.000	0.000	0.033	0.033	0.066	0.033	0.113
tai060a	0.819	0.826	0.904	0.894	0.858	0.976	0.899	0.865	0.971	0.762	0.888	0.901
tai060b	0.037	0,000	0,000	0,000	0.037	0,000	0,000	0,000	0,000	0,000	0,000	0.04
wil050	0.002	0.003	0.013	0.007	0.007	0.011	0.008	0,000	0.005	0.007	0.011	0.007
Average:	2.266	2.471	2.866	2.779	2.615	3.29	2.633	2.659	2.608	2.278	2.001	3.279

In the second group of experiments, the values of the control parameters remain the same. The obtained results are shown in Table 2.

In the third setup of the experiments, the number of tabu search iterations and the number of iterations of the hierarchical tabu search procedure are increased. In fact, 12 different configurations of the algorithm are compared. The results of the experiments are given in Table 3 and Fig. 6.

Table 2. Results of the comparison of mutation procedures

	Average relative deviation											
Mut. var.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bl049	0.730	0.800	0.994	0.712	0.739	0.642	1.099	0.976	1.02	1.082	0.624	0.792
bl064	1.009	1.075	1.336	0.855	0.862	1.049	1.229	1.376	1.229	2.057	0.755	1.336
ci049	0.004	0.004	0.080	0.021	0.000	0.001	0.097	0.003	0.003	0.000	0.001	0.001
ci064	0.092	0.085	0.218	0.089	0.074	0.092	0.187	0.256	0.11	0.083	0.075	0.049
dre042	8.77	11.204	22.984	1.466	15.026	7.016	25.916	20.524	14.346	16.335	8.063	0.000
dre056	28.785	32.431	40.829	26.538	29.705	37.459	41.197	39.576	34.494	56.814	26.206	16.777
lipa070a	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165	0.165
lipa070b	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
sko056	0.002	0.000	0.082	0.019	0.000	0.000	0.096	0.064	0.003	0.001	0.000	0.000
sko064	0.006	0.000	0.002	0.006	0.000	0.000	0.007	0.026	0.001	0.000	0.006	0.000
tai035a	0.355	0.386	0.707	0.377	0.240	0.365	0.672	0.520	0.513	0.000	0.197	0.034
tai035b	0.000	0.000	0.000	0.084	0.000	0.000	0.000	0.000	0.037	0.000	0.000	0.028
tai040a	0.483	0.501	0.797	0.508	0.487	0.52	0.771	0.652	0.699	0.337	0.448	0.289
tai040b	0.000	0.201	0.000	0.000	0.000	0.000	0.000	0.201	0.402	0.000	0.000	0.603
tai045e1	3.253	1.376	10.231	11.784	2.714	2.246	9.454	8.456	17.034	0.000	1.301	15.490
tai050a	0.810	0.718	1.193	0.876	0.813	0.846	1.064	1.044	0.899	0.601	0.737	0.487
tai050b	0.000	0.000	0.078	0.253	0,000	0.033	0.019	0.033	0.035	0,000	0.033	0.123
tai060a	0.819	0.879	1.123	0.908	0.883	0.882	1.200	1.041	0.935	0.649	0.83	0.487
tai060b	0.037	0.000	0.002	0.201	0.000	0.000	0.037	0.005	0.000	0.000	0.000	0.409
wil050	0.002	0.005	0.017	0.018	0.003	0.003	0.025	0.011	0.014	0.000	0.007	0.000
Average:	2.266	2.492	4.042	2.244	2.586	2.566	4.162	3.746	3.597	3.906	1.972	1.854

Table 3. Results of the comparison of different configurations of the algorithm

	Average relative deviation											
Config. No.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
bl049	0.440	0.466	0.466	0.396	0.475	0.484	0.396	0.44	0.484	0.396	0.484	0.466
bl064	0.615	0.608	0.568	0.541	0.548	0.528	0.574	0.635	0.508	0.521	0.648	0.581
ci049	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
ci064	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
dre042	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
dre056	5.654	1.344	0,000	1.731	4.696	1.289	0,000	4.678	0,000	2.910	0,000	2.910
sko049	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
sko056	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
sko064	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
tai020a	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
tai035a	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
tai035b	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
tai040a	0.015	0.030	0.037	0.037	0.022	0.037	0.030	0.037	0.037	0.037	0.022	0.022
tai040b	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
tai045e1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
tai050a	0.084	0.116	0.117	0.080	0.092	0.152	0.119	0.057	0.036	0.092	0.052	0.011
tai050b	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
tai060a	0.221	0.258	0.237	0.238	0.235	0.236	0.277	0.221	0.214	0.211	0.200	0.161
tai060b	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
wil050	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Average:	**0.351**	**0.141**	**0.071**	**0.151**	**0.303**	**0.136**	**0.07**	**0.303**	**0.064**	**0.208**	**0.07**	**0.208**

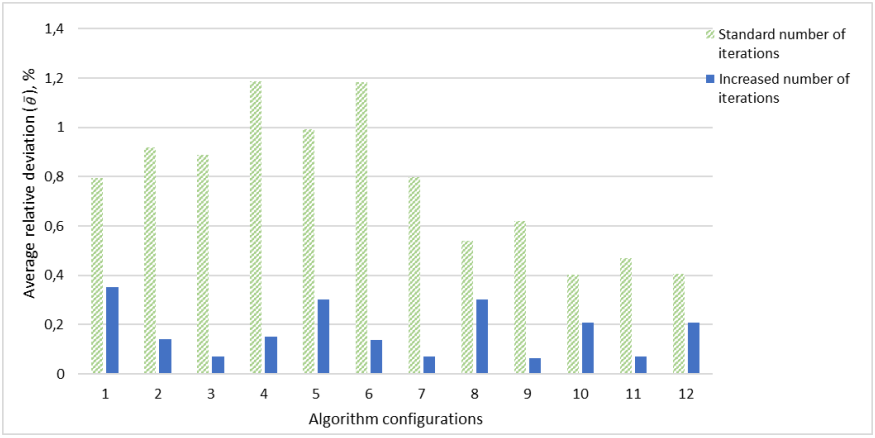


Figure 6. Results of the additional extended experiments

The main remarks regarding the results of the experiments are as follows.

The obtained results are quite influenced by the size and nature of the particular instances of the QAP. However, the differences in the effectiveness of particular crossover and mutation procedures are not statistically very significant. This is very likely due to the fact that crossovers and mutations are tested in the "hybrid environment". Still, it can be observed that the universal crossover and the exploration based mutation seem, on average, to be superior to the other procedures. The main differentiation in the results is due to the distinct configurations and parameters of the genetic algorithm itself. Not only the crossover and mutation operators, but also, for example, the number of hierarchical iterations within HITS are important (see Fig. 6).

We have also compared the GH algorithm with the memetic algorithm (MA), proposed by Benlic and Hao (Bencil, Hao, 2015), which most likely is the best so far heuristic algorithm for the QAP, based on the best available knowledge. The results of the comparison of the algorithms are presented in Tables 4, 5 and 6. It seems that the genetic-hierarchical algorithm outperforms MA. Additionally, we used the genetic algorithms by Drezner et al. (Drezner et al., 2005) and Drezner and Misevičius (Drezner, Misevičius, 2013) in the further comparison (see Tables 7, 8, 9 and 10). Again, the GH algorithm compares favourably with both the algorithms by Drezner et al. as well as Drezner and Misevičius.

Table 4. Comparative results between GHA and memetic algorithm (MA) (Bencil, Hao, 2015) (I)

Instance	BKV	GHA		MA	
		$\overline{\theta}$	Time (sec)	$\overline{\theta}$	Time (sec)
sko72	66256	**0.000**	**29.380**	**0.000**	240.000
sko81	90998	**0.000**	**95.421**	**0.000**	258.000
sko90	115534	**0.000**	**229.456**	**0.000**	918.000
sko100a	152002	**0.000**	**542.640**	**0.000**	1338.000
sko100b	153890	**0.000**	**227.774**	**0.000**	390.000
sko100c	147862	**0.000**	**400.697**	**0.000**	720.000
sko100d	149576	**0.000**	**377.108**	**0.006**	1254.000
sko100e	149150	**0.000**	**438.632**	**0.000**	714.000
sko100f	149036	**0.000**	**790.550**	**0.000**	1380.000
wil100	273038	**0.000**	**600.566**	**0.000**	870.000
tho150	8133398	**0.007**	**6300.000**	0.008	24984.000

Note. Time denotes the average CPU time per one run.

Table 5. Comparative results between GHA and memetic algorithm (MA) (Bencil, Hao, 2015) (II)

Instance	BKV	GHA		MA	
		$\overline{\theta}$	Time (sec)	$\overline{\theta}$	Time (sec)
tai40a	3139370	**0.052(3)**	**204.916**	0.059(2)	486.000
tai50a	4938796	0.192(**2**)	**268.705**	**0.131(2)**	2520.000
tai60a	7205962	0.215(1)	**713.455**	**0.144(2)**	4050.000
tai80a	13499184	**0.367**(0)	**3040.000**	0.426(0)	3948.000
tai100a	21043560	**0.311**(0)	6200.000	0.447(0)	**2646.000**

Notes. Time denotes the average CPU time per one run. In parentheses, the number of times that the BKS has been found is presented. The best known value for `tai100a` is from (Misevičius, 2019).

Table 6. Comparative results between GHA and memetic algorithm (MA) (Bencil, Hao, 2015) (III)

Instance	BKV	GHA		MA	
		$\bar{\theta}$	Time (sec)	$\bar{\theta}$	Time (sec)
tai50b	458821517	**0.000(10)**	**5.488**	**0.000(10)**	72.000
tai60b	608215054	**0.000(10)**	**5.036**	**0.000(10)**	312.000
tai80b	818415043	**0.000(10)**	**29.161**	**0.000(10)**	1878.000
tai100b	1185996137	**0.000(10)**	**83.515**	**0.000(10)**	816.000
tai150b	498896643	**0.001(4)**	**2150.000**	0.060(1)	4686.000

Note. Time denotes the average CPU time per one run.

Table 7. Comparative results between GHA and hybrid genetic algorithm (HGA) (Drezner et al., 2005) (I)

Instance	BKV	GHA		HGA	
		$\bar{\theta}$	Time (sec)	$\bar{\theta}$	Time (sec)
dre30	508	**0.000(10)**	**0.875**	**0.000**	143.400
dre42	764	**0.000(10)**	**9.809**	1.340	547.800
dre56	1086	**0.000(10)**	**86.024**	17.460	1810.800
dre72	1452	**0.000(10)**	**489.877**	27.280	5591.400
dre90	1838	**0.000(10)**	**4546.978**	33.880	11557.800
dre110	2264	**0.000(10)**	**16500.000**		
dre132	2744	**13.620(3)**	**30000.000**		

Note. Time denotes the average CPU time per one run. In parentheses, the number of times that the BKS has been found is presented.

Table 8. Comparative results between GHA and hybrid genetic algorithm (HGA) (Drezner et al., 2005) (II)

Instance	BKV	GHA		HGA	
		$\overline{\theta}$	Time (sec)	$\overline{\theta}$	Time (sec)
tai27e1	2558	**0.000**	**0.332**	**0.000**	~60.000
tai27e2	2850	**0.000**	**0.399**	**0.000**	~60.000
tai27e3	3258	**0.000**	**0.078**	**0.000**	~60.000
tai45e1	6412	**0.000**	**1.346**	**0.000**	~300.000
tai45e2	5734	**0.000**	**5.713**	**0.000**	~300.000
tai45e3	7438	**0.000**	**2.471**	**0.000**	~300.000
tai75e1	14488	**0.000**	**52.287**	**0.000**	~2220.000
tai75e2	14444	**0.000**	**25.134**	0.339	~2220.000
tai75e3	14154	**0.000**	**36.677**	**0.000**	~2220.000

Note. Time denotes the average CPU time per one run.

Table 9. Comparative results between GHA and hybrid genetic algorithm with differential improvement (HGA-DI) (Drezner, Misevičius, 2013) (I)

Instance	BKV	GHA		HGA-DI	
		$\overline{\theta}$	Time (sec)	$\overline{\theta}$	Time (sec)
bl36	3296	**0.000(10)**	**9.491**	**0.000(10)**	51.000
bl49	4548	**0.229(3)**	230.746	0.334(0)	**125.000**
bl64	5988	**0.207(2)**	519.594	0.227(0)	**356.000**
bl81	7532	**0.319(1)**	1291.861	0.494(0)	**937.000**
bl100	9264	**0.479(1)**	**1963.669**	0.548(0)	2306.000
bl121	11400	**0.516(** 0)	22000.000	0.867(0)	**5510.000**
bl144	13460	**0.431(** 0)	43000.000	0.993(0)	**12490.000**

Notes. Time denotes the average CPU time per one run. In parentheses, the number of times that the BKS has been found is presented.

Table 10. Comparative results between GHA and hybrid genetic algorithm with differential improvement (HGA-DI) (Drezner, Misevičius, 2013) (II)

Instance	BKV	GHA		HGA-DI	
		$\overline{\theta}$	Time (sec)	$\overline{\theta}$	Time (sec)
ci36	168611971	**0.000**(**10**)	**1.279**	**0.000**(**10**)	50.000
ci49	236355034	**0.000**(**10**)	**6.864**	**0.000**(**10**)	124.000
ci64	325671035	**0.000**(**10**)	**46.818**	**0.000**(**10**)	354.000
ci81	427447820	**0.000**(**10**)	**250.470**	**0.000**(**10**)	932.000
ci100	523146366	**0.000**(**10**)	9998.000	0.007(3)	**2285.000**
ci121	653409588	**0.026**(**2**)	52000.000	0.051(1)	**5431.000**
ci144	794811636	**0.050**(**1**)	80000.000	0.068(**1**)	**12430.000**

Notes. Time denotes the average CPU time per one run. In parentheses, the number of times that the BKS has been found is presented.

CONCLUSIONS

1. The analysis and conceptual comparison of the existing heuristic algorithms has shown that the use of population-based evolutionary and iterative algorithms is an effective way to cope with the solution of difficult combinatorial tasks. Based on this, the hierarchical population-based genetic algorithm is chosen to be investigated in this work.
2. In particular, a genetic hierarchical algorithm (GHA) for the solution of the quadratic assignment problem is proposed. The most essential contribution and novelty is that the GHA makes use of the innovative genetic crossover procedures; in addition, GHA incorporates the hierarchical iterated tabu search algorithm, which in turn includes the effective mutation procedures to diversify the search process and discover new regions in the search space.
3. The obtained results of the computational experiments demonstrate the high performance of GHA in terms of the solution quality and the CPU time of the algorithm. The GH algorithm with the tuned ("tweaked") integrated hierarchical tabu search and ad hoc crossover and mutation procedures is deemed to be superior to other heuristic algorithms for the examined data instances of the QAP.
4. Still, it is recommended that the further additional computational experiments should be performed in order to learn more about which of the configurations of GHA (including the identification of the most satisfiable number of the hierarchical iterations and levels of HITS) is the best.

7 LITERATŪRA IR ŠALTINIAI

1. AARTS, E.H.L.; KORST, J.H.M. (1989). *Simulated Annealing and Boltzmann Machines*, Chichester: Wiley.

2. AARTS, E.H.L.; KORST, J.H.M. (2001). Selected topics in simulated annealing. In RIBEIRO, C.C.; HANSEN, P. (eds.), *Essays and Surveys in Metaheuristics*, Boston: Kluwer, 1-37.

3. AARTS, E.H.L.; LENSTRA, J.K. (eds.) (1997). *Local Search in Combinatorial Optimization*, Chichester: Wiley.

4. ABDEL-BASSET, M.; MANOGARAN, G.; EL-SHAHAT, D.; MIRJALILI, S. (2018). Integrating the whale algorithm with tabu search for quadratic assignment problem: a new approach for locating hospital departments. *Applied Soft Computing*, Vol.73, 530-546.

5. ABDEL-BASET, M.; WU, H.; ZHOU, Y.; ABDEL-FATAH, L. (2017). Elite opposition-flower pollination algorithm for quadratic assignment problem. *Journal of Intelligent & Fuzzy Systems*, Vol.33, 901-911.

6. ABDEL-BASSET, M.; MANOGARAN, G.; RASHAD, H.; ZAIED, A.N.H. (2018). A comprehensive review of quadratic assignment problem: variants, hybrids and applications. *Journal of Ambient Intelligence and Humanized Computing*, 1-24.

7. ABDELKAFI, O.; DERBEL, B.; LIEFOOGHE, A. (2019). A parallel tabu search for the large-scale quadratic assignment problem. *IEEE Congress on Evolutionary Computation, IEEE CEC 2019*, Piscataway: IEEE, 3070-3077.

8. ACAN, A.; ÜNVEREN, A. (2015). A great deluge and tabu search hybrid with two-stage memory support for quadratic assignment problem. *Applied Soft Computing*, 2015, Vol.36, 185-203.

9. ACHARY, T.; PILLAY, S.; PILLAI, S.M.; MQADI, M.; GENDERS, E.; EZUGWU, A.E. (2021). A performance study of meta-heuristic approaches for quadratic assignment problem. *Concurrency and Computation. Practice and Experience*, Vol.33. https://doi.org/10.1002/cpe.6321

10. AHMED, Z.H. (2015). A multi-parent genetic algorithm for the quadratic assignment problem. *OPSEARCH*, Vol.52, 714-732.

11. AHMED, Z.H. (2016). Experimental analysis of crossover and mutation operators on the quadratic assignment problem. *Annals of Operations Research*, Vol.247, 833-851.

12. AHMED, Z.H. (2018). A hybrid algorithm combining lexisearch and genetic algorithms for the quadratic assignment problem. *Cogent Engineering*, Vol.5, 1423743.

13. AHMED, A.K.M.F.; SUN, J.U. (2018). A novel approach to combine the hierarchical and iterative techniques for solving capacitated location-routing problem. *Cogent Engineering*, Vol.5, 1463596.

14. AHUJA, R.K.; ORLIN, J.B.; TIWARI, A. (2000). A greedy genetic algorithm for the quadratic assignment problem. *Computers & Operations Research*, Vol.27, 917-934.

15. AKSAN, Y.; DOKEROGLU, T.; COSAR, A. (2017). A stagnation-aware cooperative parallel breakout local search algorithm for the quadratic assignment problem. *Computers & Industrial Engineering*, Vol.103, 105-115.

16. ANGEL, E.; ZISSIMOPOULOS, V. (1998). On the quality of local search for the quadratic assignment problem. *Discrete Applied Mathematics*, Vol.82, 15-25.

17. ANSTREICHER, K.M.; BRIXIUS, N.W. (2001). A new bound for the quadratic assignment problem based on convex quadratic programming. *Mathematical Programming*, Vol.89, 341-357.

18. ANSTREICHER, K.M.; BRIXIUS, N.W.; GAUX, J.P.; LINDEROTH, J. (2002) Solving large quadratic assignment problems on computational grids. *Mathematical Programming*, Vol.91, 563-588.

19. ARMOUR, G.C.; BUFFA, E.S. (1963). A heuristic algorithm and simulation approach to relative location of facilities. *Management Science*, Vol.9, 294-304.

20. BALDÉ, M.A.M.T.; GUEYE, S.; NDIAYE, B.M. (2020). A greedy evolutionary hybridization algorithm for the optimal network and quadratic assignment problem. *Operational Research*.

21. BATTARRA, M.; BENEDETTINI, S.; ROLI, A. (2011). Leveraging saving-based algorithms by master–slave genetic algorithms. *Engineering Applications of Artificial Intelligence*, Vol.24, 555-566.

22. BATTITI, R.; TECCHIOLLI, G. (1994). The reactive tabu search. *ORSA Journal on Computing*, Vol.6, 126-140.

23. BAUM, E.B. (1986). Towards practical "neural" computation for combinatorial optimization problems. In DENKER, J.S. (ed.), *Neural Networks for Computing*, New York: American Institute of Physics, 53-58.

24. BEAN, J.C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, Vol.6, 154-160.

25. BEN-DAVID, G.; MALAH, D. (2005). Bounds on the performance of vector-quantizers under channel errors. *IEEE Transactions on Information Theory*, Vol.51, 2227-2235.

26. BENLIC, U.; HAO, J.-K. (2013). Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation*, Vol.219, 4800-4815.

27. BENLIC U.; HAO, J.-K. (2015). Memetic search for the quadratic assignment problem. *Expert Systems with Applications*, Vol.42, 584-595.

28. BLUM, C.; ROLI, A. (2003). Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM *Computing Surveys*, Vol.35, 268-308.

29. BOUSSAÏD, I.; LEPAGNOT, J.; SIARRY, P. (2013). A survey on optimization metaheuristics. *Information Sciences*, Vol.237, 82-117.

30. BÖLTE, A.; THONEMANN, U.W. (1996). Optimizing simulated annealing schedules with genetic programming. *European Journal of Operational Research*, Vol.92, 402-416.

31. BROWNLEE, J. (2011). *Clever Algorithms. Nature-Inspired Programming Recipes*. lulu.com.

32. BRUSCO, M.J.; STAHL, S. (2000). Using quadratic assignment methods to generate initial permutations for least-squares unidimensional scaling of symmetric proximity matrices. *Journal of Classification*, Vol.17, 197-223.

33. BUFFA, E.S.; ARMOUR, G.C.; VOLLMANN, T.E. (1964). Allocating facilities with CRAFT. *Harvard Business Review*, Vol.42, 136-158.

34. BURKARD, R.E. (1984). Quadratic assignment problems. *European Journal of Operational Research*, Vol.15, 283-289.

35. BURKARD, R.E. (1991). Locations with spatial interactions: the quadratic assignment problem. In MIRCHANDANI, P.B.; FRANCIS, R.L. (eds.), *Discrete Location Theory*, New York: Wiley, 387-437.

36. BURKARD, R.E.; OFFERMANN, J. (1977). Entwurf von schreibmaschinentastaturen mittels quadratischer zuordnungsprobleme. *Zeitschrift für Operations Research*, 1977, Vol.21, 121-132.

37. BURKARD, R.E.; ÇELA, E.; PARDALOS, P.M.; PITSOULIS, L. (1998). The quadratic assignment problem. DU, D.Z.; PARDALOS, P.M. (eds.), *Handbook of Combinatorial Optimization*. Kluwer, Vol.3, 241-337.

38. BURKARD, R.; KARISCH, S.; RENDL, F. (1997). QAPLIB — a quadratic assignment problem library. *Journal of Global Optimization*, Vol.10, 391-403. [Žr. taip pat prieigą per internetą: <http://anjos.mgi.polymtl.ca/qaplib/>.]

39. BURKARD, R.E.; RENDL, F. (1984). A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operational Research*, Vol.17, 169-174.

40. CÁRDENAS, E.G.; POVEDA, R.CH.; GARCÍA, O.H. (2017). A solution for the quadratic assignment problem (QAP) through a parallel genetic algorithm based grid on GPU. *Applied Mathematical Sciences*, Vol.11, 2843-2854.

41. CARUANA, R.A.; ESHELMAN, L.A.; SCHAFFER, J.D. (1989). Representation and hidden bias II: Eliminating defining length bias in genetic search via shuffle crossover. Proceedings of the Eleventh International Joint Conference on AI, SRIDHARAN, N.S. (ed.), Vol.1, San Mateo: Morgan Kaufmann, 750-755.

42. CERNÝ, V. (1982). A thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. Tech. Report, Comenius University, Bratislava, CSSR.

43. ÇELA, E. (1998). *The Quadratic Assignment Problem: Theory and Algorithms*. Dordrecht: Kluwer.

44. CHMIEL, W. (2019). Evolutionary algorithm using conditional expectation value for quadratic assignment problem. *Swarm and Evolutionary Computation*, Vol.46, 1-27.

45. CHMIEL, W.; KWIECIEŃ, J. (2018). Quantum-inspired evolutionary approach for the quadratic assignment problem. *Entropy*, Vol.20, 781.

46. CONNOLLY, D.T. (1990). An improved annealing scheme for the QAP. *European Journal of Operational Research*, Vol.46, 93-100.

47. CZAPIŃSKI, M. (2013). An effective parallel multistart tabu search for quadratic assignment problem on CUDA platform. *Journal of Parallel and Distributed Computing*, Vol.73, 1461-1468.

48. DANTAS, A.; POZO, A. (2020). On the use of fitness landscape features in meta-learning based algorithm selection for the quadratic assignment problem. *Theoretical Computer Science*, Vol.805, 62–75.

49. DATE, K.; NAGI, R. (2019). Level 2 reformulation linearization technique–based parallel algorithms for solving large quadratic assignment problems on graphics processing unit clusters. *INFORMS Journal on Computing*, Vol.31, 771-789.

50. DE CARVALHO, JR., S.A.; RAHMANN, S. (2006). Microarray layout as a quadratic assignment problem. In HUSON, D.; KOHLBACHER, O.; LUPAS, A.; NIESELT, K.; ZELL, A. (eds.), *Proceedings of the German Conference on Bioinformatics*, Vol.83, Bonn: Gesellschaft für Informatik, 11-20.

51. DELL'AMICO, M.; DÍAZ, J.C.D.; IORI, M.; MONTANARI, R. (2009). The single-finger keyboard layout problem. *Computers & Operations Research*, Vol.36, 3002-3012.

52. DELL'AMICO, M.; TRUBIAN, M. (1998). Solution of large weighted equicut problems. *European Journal of Operational Research*, Vol.106, 500-521.

53. DEN BESTEN, M.; STÜTZLE, T; DORIGO, M. (2001). Design of iterated local search algorithms. An example application to the single machine total weighted tardiness problem. In BOERS, E.J.W. et al. (eds.), *Applications of Evolutionary Computing, EvoWorkshops 2001, Lecture Notes in Computer Science*, Vol.2037, Berlin, Heidelberg: Springer, 441-451.

54. DICKEY, J.W.; HOPKINS, J.W. (1972). Campus building arrangement using TOPAZ. *Transportation Research*, Vol.6, 59-68.

55. DOKEROGLU, T. (2015). Hybrid teaching–learning-based optimization algorithms for the quadratic assignment problem. *Computers & Industrial Engineering*, Vol.85, 86-101.

56. DOKEROGLU, T.; COSAR, A. (2016). A novel multistart hyper-heuristic algorithm on the grid for the quadratic assignment problem. *Engineering Applications of Artificial Intelligence*, Vol.52, 10-25.

57. DOKEROGLU, T.; SEVINC, E.; COSAR, A. (2019). Artificial bee colony optimization for the quadratic assignment problem. *Applied Soft Computing*, Vol.76, 595-606.

58. DREZNER, Z. (2003). A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing*, Vol.15, 320-330.

59. DREZNER, Z. (2006). Finding a cluster of points and the grey pattern quadratic assignment problem. *OR Spectrum*, Vol.28, 417-436.

60. DREZNER, Z. (2015). The quadratic assignment problem. In LAPORTE, G.; NICKEL, S.; SALDANHA DA GAMA, F. (eds.), *Location Science*, Cham: Springer International Publishing, 345-363.

61. DREZNER, Z. (2019). Taking advantage of symmetry in some quadratic assignment problems. *INFOR: Information Systems and Operational Research*, Vol.57, 623 641.

62. DREZNER, Z.; DREZNER, T.D. (2019). The alpha male genetic algorithm. *IMA Journal of Management Mathematics*, Vol.30, 37–50.

63. DREZNER, Z.; DREZNER, T.D. (2020). Biologically inspired parent selection in genetic algorithms. *Annals of Operations Research*, Vol.287, 161–183.

64. DREZNER, Z.; HAHN, P.M.; TAILLARD, E.D. (2005). Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for metaheuristic methods. *Annals of Operations Research*, Vol.139, 65-94.

65. DREZNER, Z.; MISEVIČIUS, A. (2013). Enhancing the performance of hybrid genetic algorithms by differential improvement. *Computers & Operations Research*, Vol.40, 1038-1046.

66. DRIRA, A.; PIERREVAL, H.; HAJRI-GABOUJ, S. (2007). Facility layout problems: a survey. *Annual Reviews in Control*, 2007, Vol.31, 255-267.

67. DU, D.Z.; PARDALOS, P.M.; GRAHAM, R.L. (eds.). (2013). *Handbook of Combinatorial Optimization*, Berlin-Heidelberg: Springer.

68. DUMAN, E.; UYSAL, M.; ALKAYA, A.F. (2012). Migrating birds optimization: a new metaheuristic approach and its performance on quadratic assignment problem. *Information Sciences*, Vol.217, 65-77.

69. EIBEN, A.E.; RAUE, P.-E.; RUTTKAY, Z. (1994). Genetic algorithms with multi-parent recombination. Parallel Problem Solving fro*m Nature — PPSN III, International Conference on Evolutionary Computation*, The Third Conference on Parallel Problem Solving from Nature, Proceedings, DAVIDOR, Y.; SCHWEFEL, H.P.; MÄNNER, R. (eds.), *Lecture Notes in Computer Science*, Vol.866, Berlin-Heidelber: Springer, 78-87.

70. ELSHAFEI, A.N. (1977). Hospital layout as a quadratic assignment problem. *Operations Research Quarterly*, Vol.28, 167-179.

71. FERREIRA, J.F.S.B.; KHOO, Y.; SINGER, A. (2018). Semidefinite programming approach for the quadratic assignment problem with a sparse graph. *Computational Optimization and Applications*, Vol.69, 677-712.

72. FRANCIS, R.L.; WHITE, J.A. (1998). *Facility Layout and Location: An Analytical Approach*. Prentice Hall: Englewood Cliffs.

73. FRIEZE, A.M.; YADEGAR, J.; EL-HORBATY, S.; PARKINSON, D. (1989). Algorithms for assignment problems on an array processor. *Parallel Computing*, Vol.11, 151-162.

74. GAMBARDELLA, L.M.; TAILLARD, E.; DORIGO, M. (1999). Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, Vol.50, 167-176.

75. GARAI, G.; CHAUDHURI, B.B. (2007). A distributed hierarchical genetic algorithm for efficient optimization and pattern matching. *Pattern Recognition*, Vol.40, 212–228. https://doi.org/10.1016/j.patcog.2006.04.023

76. GAREY, M.R.; JOHNSON, D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: Freeman.

77. GILMORE, P.C. (1962). Optimal and suboptimal algorithms for the quadratic assignment problem. *SIAM Journal of Applied Mathematics*, Vol.14, 305-313.

78. GLOVER, F. (1989). Tabu search: part I. *ORSA Journal on Computing*, Vol.1, 190-206.

79. GLOVER, F. (1990). Tabu search: part II. *ORSA Journal on Computing*, Vol.2, 4-32.

80. GLOVER, F. (1994). Genetic algorithms and scatter search: unsuspected potential. *Statistics and Computing*, 1994, Vol.4, 131-140.

81. GLOVER, F.; LAGUNA, M. (1997). *Tabu Search*. Dordrecht: Kluwer.

82. GOLDBERG, D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading: Addison-Wesley.

83. GOLDBERG, D.E.; LINGLE, R. (1985). Alleles, loci, and the traveling salesman problem. Proceedings of the First International Conference on Genetic Algorithms and their Applications, GREFENSTETTE, J.J. (ed.), Hillsdale: Lowrence Erlbaum, 154-159.

84. GONÇALVES, A.D.; PESSOA, A.A.; BENTES, C.; FARIAS, R.; DE DRUMMOND, A.L.M. (2017). A graphics processing unit algorithm to solve the quadratic assignment problem using level-2 reformulation-linearization technique. *INFORMS Journal on Computing*, Vol.29, 676-687.

85. HAFIZ, F.; ABDENNOUR, A. (2016). Particle swarm algorithm variants for the quadratic assignment problems - a probabilistic learning approach. *Expert Systems with Applications*, Vol.44, 413-431.

86. HAHN, P.; GRANT, T. (1998). Lower bounds for the quadratic assignment problem based upon a dual formulation. *Operations Research*, Vol.46. 912–922.

87. HAHN, P.M.; KRARUP, J. (2001). A hospital facility problem finally solved. *Journal of Intelligent Manufacturing*, Vol.12, 487-496.

88. HAHN, P.M.; ZHU, Y.-R.; GUIGNARD, M.; HIGHTOWER, W.L.; SALTZMAN, M.J. (2012). A level-3 reformulation-linearization technique-based bound for the quadratic assignment problem. *INFORMS Journal on Computing*, Vol.24, 202-209.

89. HANAN, M.; KURTZBERG, J.M. (1972). Placement techniques. In BREUER, M.A. (ed.), *Design Automation of Digital Systems: Theory and Techniques*, Vol.1, Englwood Cliffs: Prentice-Hall, 213–282.

90. HANSEN, P.; JAUMARD, B. (1987). Algorithms for the maximum satisfiability problem. RUTCOR Research Report 43-87, Rutgers University, USA.

91. HARRIS, M.; BERRETTA, R.; INOSTROZA-PONTA, M.; MOSCATO, P. (2015). A memetic algorithm for the quadratic assignment problem with parallel local search. *2015 IEEE Congress on Evolutionary Computation (CEC), Sendai, Japan, Proceedings*, Piscataway: IEEE, 838-845.

92. HAUPT, R.L.; HAUPT, S.E. (2004). *Practical Genetic Algorithms*. Hoboken: Wiley-Interscience.

93. HAUSCHILD, M.; BHATIA, S.; PELIKAN, M. (2012). Image segmentation using a genetic algorithm and hierarchical local search. In SOULE, T. (ed.), *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation* (Philadelphia, USA), New York: ACM Press, 633-639.

94. HEFFLEY, D.R. (1977). Assigning runners to a relay team. In LADANY, S.P.; MACHOL, R.E. (eds.). *Optimal Strategies in Sports*. Amsterdam: North-Holland, 169-171.

95. HELAL, A.M.; ABDELBAR, A.M. (2017). Incorporating domain-specific heuristics in a particle swarm optimization approach to the quadratic assignment problem. *Memetic Computing*, Vol.6, 241-254.

96. HERRERA, F.; LOZANO, M.; SÁNCHEZ, A.M. (2003). A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. *International Journal of Intelligent Systems*, Vol.18, 309-338.

97. HERTHEL, A.B.; SUBRAMANIAN, A. (2020). Optimizing single-finger keyboard layouts on smartphones. *Computers & Operations Research*, Vol.120, 104947.

98. HOLLAND, J.H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press.

99. HOOS, H.H.; STÜTZLE, T. (2004). *Stochastic Local Search. Foundations and Applications.* San Francisco: Morgan Kaufmann.

100. HUSSAIN, K.; SALLEH, M.N.M.; CHENG, S.; SHI, Y. (2019). Metaheuristic research: a comprehensive survey. *Artificial Intelligence Review*, Vol.52, 2191-2233.

101. HUSSIN, M.S., STÜTZLE, T. (2009). Hierarchical iterated local search for the quadratic assignment problem. In BLESA, M.J., BLUM, C., DI GASPERO, L., ROLI, A., SAMPELS, M., SCHAERF, A. (eds.), *Hybrid Metaheuristics, HM 2009, Lecture Notes in Computer Science*, Vol.5818, Berlin-Heidelberg: Springer, 115-129.

102. ISMAIL, M.M.; HEZAM, I.M.; EL-SHARKAWY, E. (2017). Enhanced cuckoo search algorithm with SPV rule for quadratic assignment problem. *International Journal of Computer Applications*, Vol.158, 39-42.

103. KILIÇ, H.; YÜZGEÇ, U. (2019). Tournament selection based antlion optimization algorithm for solving quadratic assignment problem. *Engineering Science and Technology, an International Journal*, Vol.22, 673-691.

104. KIRKPATRICK, S.; GELATT, C.D., JR., VECCHI, M.P. (1983). Optimization by simulated annealing. *Science*, Vol.220, p. 671-680.

105. KOOPMANS, T.; BECKMANN, M. (1957). Assignment problems and the location of economic activities. *Econometrica*, Vol.25, 53-76.

106. LALLA-RUIZ, E.; EXPÓSITO-IZQUIERDO, C.; MELIÁN-BATISTA, B.; MORENO-VEGA, M.J. (2016). A hybrid biased random key genetic algorithm for the quadratic assignment problem. *Information Processing Letters*, Vol.116, 513-520.

107. LAPORTE, G.; MERCURE, H. (1988). Balancing hydraulic turbine runners: a quadratic assignment problem. *European Journal of Operational Research*, Vol.35, 378-381.

108. LAWLER, E.L. (1973). Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, Vol.19, 544–546.

109. LI, Y.; PARDALOS, P.M.; RESENDE, M.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, 237-261.

110. LIM, M.H.; YUAN, Y.; OMATU, S. (2000). Efficient genetic algorithms using simple genes exchange local search policy for the quadratic assignment problem. *Computational Optimization and Applications*, Vol.15, 249-268.

111. LIM, S.M.; SULTAN, A.B.MD.; SULAIMAN, MD.N.; MUSTAPHA, A.; LEONG, K.Y. (2017). Crossover and mutation operators of genetic algorithms. *International Journal of Machine Learning and Computing*, Vol.7, 9-12.

112. LIM, W.L.; WIBOWO, A.; DESA, M.I.; HARON, H. (2016). A biogeography-based optimization algorithm hybridized with tabu search for the quadratic assignment problem. *Computational Intelligence and Neuroscience*, Vol.2016, 1-12.

113. LOIOLA, E.M.; DE ABREU, N.M.M.; BOAVENTURA-NETTO, P.O.; HAHN, P.; QUERIDO, T. (2007). A survey for the quadratic assignment problem. *European Journal of Operational Research*, Vol.176, 657-690.

114. LOURENCO, H.R.; MARTIN, O.; STÜTZLE, T. (2002). Iterated local search. In GLOVER, F.; KOCHENBERGER, G. (eds.), *Handbook of Metaheuristics*. Norwell: Kluwer, 321-353.

115. LSTIBŮREK, M.; STEJSKAL, J.; MISEVIČIUS, A.; KORECKÝ, J.; EL-KASSABY, Y. (2015). Expansion of the minimum-inbreeding seed orchard design to operational scale. *Tree Genetics & Genomes*, Vol.11, 1-8.

116. MARTIN, O.; OTTO, S.W. (1996). Combining simulated annealing with local search heuristics. *Annals of Operations Research*, Vol.63, 57-75.

117. MARTIN, O.; OTTO, S.W.; FELTEN, E.W. (1992). Large-step Markov chains for the TSP incorporating local search heuristics. *Operations Research Letters*, Vol.11, 219-224.

118. MÁXIMO, V.R.; NASCIMENTO, M.C.V. (2019). Intensification, learning and diversification in a hybrid metaheuristic: an efficient unification. *Journal of Heuristics*, Vol.25, 539-564.

119. MERZ, P.; FREISLEBEN, B. (1997). A genetic local search approach for the quadratic assignment problem. In BÄCK, T. (ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms*, Morgan Kaufmann, East Lansing, 465-472.

120. MERZ, P.; FREISLEBEN, B. (2000). Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, Vol.4, 337-352.

121. METROPOLIS, N.; ROSENBLUTH, A.; ROSENBLUTH, M.; TELLER, A.; E.TELLER. (1953). Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, Vol.21, 1087-1092.

122. MICHALEWICZ, Z.; FOGEL, D.B. (2000). How to Solve It: Modern Heuristics. Berlin-Heidelberg: Springer.

123. MIHIC, K.; RYAN, K.; WOOD, A. (2018). Randomized decomposition solver with the quadratic assignment problem as a case study. *INFORMS Journal on Computing*, Vol.30, 295-308.

124. MILINIS, A. (2005). Hibridinis genetinis algoritmas ir jo modifikacijos kvadratinio pasiskirstymo uždaviniui spręsti. Magistro tezės, Kaunas: Technologija. [Žr. taip pat prieigą per internetą: <http://talpykla.elaba.lt/elaba-fedora/objects/elaba:1777936/datastreams/ATTACHMENT_1777939/content>.]

125. MISEVIČIUS, A. (2003). A modified simulated annealing algorithm for the quadratic assignment problem. Informatica, Vol.14, 497-514.

126. MISEVICIUS, A. (2004). An improved hybrid genetic algorithm: new results for the quadratic assignment problem. *Knowledge-Based Systems*, Vol.17, 65-73.

127. MISEVICIUS, A. (2005). A tabu search algorithm for the quadratic assignment problem. *Computational Optimization and Applications*, Vol.30, 95-111.

128. MISEVIČIUS, A. (2006). Testing of crossover operators for the grey pattern problem. Ūkio technologinis ir ekonominis vystymas (Technological and Economic Development of Economy), T.12, Nr.1, 37-43.

129. MISEVIČIUS, A. (2008). Kryžminimo procedūros kvadratinio paskirstymo uždaviniui. 14th International Conference on Information and Software Technologies, IT-2008, Research Communications, Kaunas: Technologija, 11-16.

130. MISEVICIUS, A. (2008). Restart-based genetic algorithm for the quadratic assignment problem. In BRAMER, M.; COENEN, F.; PETRIDIS, M. (eds.), *Research and Development in Intelligent Systems, Proceedings of AI-2008, the Twenty-eighth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, London: Springer, 91-104.

131. MISEVICIUS, A. (2012). An implementation of the iterated tabu search algorithm for the quadratic assignment problem. *OR Spectrum*, Vol.34, 665-690.

132. MISEVIČIUS, A. (2019). Letter: New best known solution for the most difficult QAP instance "tai100a". Memetic Computing, Vol.11, 331–332. https://doi.org/10.1007/s12293-019-00289-y

133. MISEVIČIUS, A.; PALUBECKIS, G.; DREZNER, Z. (2021). Hierarchicity-based (self-similar) hybrid genetic algorithm for the grey pattern quadratic assignment problem. Memetic Computing, Vol.13, 69-90. https://doi.org/10.1007/s12293-020-00321-6

134. MISEVIČIUS, A.; RUBLIAUSKAS, D. (2009). Testing of hybrid genetic algorithms for structured quadratic assignment problems. *Informatica*, Vol.20, 255-272.

135. MOSCATO, P. (1999). Memetic algorithms: a short introduction. In CORNE, D.; DORIGO, M.; GLOVER, F. (eds.), *New Ideas in Optimization*, London: McGraw-Hill, 219-234.

136. MURTAGH, B.A.; JEFFERSON, T.R.; SORNPRASIT, V. (1982). A heuristic procedure for solving the quadratic assignment problem. European Journal of Operational Research, Vol.9, 71-76.

137. MURTHY, K.A.; LI, Y.; PARDALOS, P.M. (1992). A local search algorithm for the quadratic assignment problem. *Informatica*, Vol.3, 524-538.

138. MZILI, I.; RIFFI, M.E.; BENZEKRI, F. (2017). Penguins search optimization algorithm to solve quadratic assignment problem. *Proceedings of the 2nd International Conference on Big Data, Cloud and Applica*tions, New York: ACM, 1-6.

139. NG, K.M.; TRAN, T.H. (2019). A parallel water flow algorithm with local search for solving the quadratic assignment problem. *Journal of Industrial and Management Optimization*, Vol.15, 235-259.

140. NUGENT, C.E.; VOLLMANN, T.E.; RUML, J. (1968). An experimental comparison of techniques for the assignment of facilities to locations. *Journal of Operations Research*, Vol.16, 150-173.

141. NYBERG, A.; WESTERLUND, T. (2012). A new exact discrete linear reformulation of the quadratic assignment problem. *European Journal of Operational Research*, Vol.220, 314-319.

142. NYSTRÖM, M. (1999). Solving certain large instances of the quadratic assignment problem: Steinberg's examples. Tech. Rep., California Institute of Technology, USA, Working Paper.

143. OLIVER, I.M.; SMITH, D.J.; HOLLAND, J.H. (1987). A study of permutation crossover operators on the traveling salesman problem. Genetic Algorithms and their

Applications: Proceedings of the Second International Conference on Genetic Algorithms and their Applications, GREFENSTETTE, J.J. (ed.), Hillsdale: Lawrence Erlbaum, 224-230.

144. OSMAN, I.H.; KELLY, J.P. (eds.). (1996). *Meta-Heuristics: Theory and Applications*. Boston-London-Dordrecht: Kluwer.

145. ÖZÇETİN, E.; ÖZTÜRK, G. (2016). A hybrid genetic algorithm for the quadratic assignment problem on graphics processing units. *Anadolu University Journal of Science and Technology A - Applied Sciences and Engineering*, Vol.17, 167-180.

146. PARDALOS, P.M.; MURTHY, K.A.; HARRISON, T.P. (1993). A computational comparison of local search heuristics for solving quadratic assigment problems. *Informatica*, Vol.4, 172-187.

147. PAUL, G. (2011). An efficient implementation of the simulated annealing heuristic for the quadratic assignment problem. Computing Research Repository, arXiv:1111.1353.

148. PAVAI, G.; GEETHA, T.V. (2017). A survey on crossover operators. *ACM Computing Surveys*, Vol.49, 1-43.

149. PHILLIPS, A.T.; ROSEN, J.B. (1994). A quadratic assignment formulation of the molecular conformation problem. *Journal of Global Optimization*, Vol.4, 229-241.

150. QAWQZEH, Y.K.; JARADAT, G.; AL-YOUSEF, A.; ABU-HAMDAH, A.; ALMARASHDEH, I.; ALSMADI, M.; TAYFOUR, M.; SHAKER, K.; HADDAD, F. (2020). Applying the big bang-big crunch metaheuristic to large-sized operational problems. *International Journal of Electrical and Computer Engineering*, Vol.10, 2484-2502.

151. RAMKUMAR, A.S.; PONNAMBALAM, S.G.; JAWAHAR, N.; SURESH, R.K. (2008). Iterated fast local search algorithm for solving quadratic assignment problems. *Robotics and Computer-Integrated Manufacturing*, Vol.24, 392-401.

152. REEVES, C.R.; ROWE, J.E. (2001). *Genetic Algorithms: Principles and Perspectives*, Norwell: Kluwer.

153. RENDL, F.; SOTIROV, R. (2007). Bounds for the quadratic assignment problem using the bundle method. *Mathematical Programming*, Vol.109, 505-524.

154. RIBEIRO, C.C.; HANSEN, P. (eds.). (2001). *Essays and Surveys in Metaheuristics*. Boston: Kluwer.

155. RODRIGUEZ, J.M.; MACPHEE, F.C.; BONHAM, D.J.; BHAVSAR, V.C. (2004). Solving the quadratic assignment and dynamic plant layout problems using a new hybrid meta-heuristic approach. In ESKICIOGLU, M.R. (ed.), *Proceedings of the 18th Annual International Symposium on High Performance Computing Systems and Applications (HPCS)*, Department of Computer Science, Winnipeg, Canada, 9-16.

156. ROKBANI, N.; KROMER, P.; TWIR, I.; ALIMI, A.M. (2020). A hybrid hierarchical heuristic-ACO with local search applied to travelling salesman problem. *International Journal of System Dynamics Applications*, Vol.9, 58-73. https://doi.org/10.4018/IJSDA.2020070104

157. ROUPIN, F. (2004). From linear to semidefinite programming: an algorithm to obtain semidefinite relaxations for bivalent quadratic problems. *Journal of Combinatorial Optimization*, Vol.8, 469-493.

158. SAHNI, S.; GONZALEZ, T. (1976). P-complete approximation problems. *Journal of ACM*, Vol.23, 555-565.

159. SALHI, S. (1998). Heuristic search methods. In MARCOULIDES, G. (ed.), *Modern Methods for Business Research*. Mahwah: Lawrence Erlbaum, 147-175.

160. SAMANTA, S.; PHILIP, D.; CHAKRABORTY, S. (2019). A quick convergent artificial bee colony algorithm for solving quadratic assignment problems. *Computers & Industrial Engineering*, Vol.137, 106070.

161. SCHAEFER, R.; BYRSKI, A.; KOŁODZIEJ, J.; SMOŁKA, M. (2012). An agent-based model of hierarchic genetic search. *Computers & Mathematics with Applications*, Vol.64, 3763–3776. https://doi.org/10.1016/j.camwa.2012.02.052

162. SCHRIMPF, G.; SCHNEIDER, J.; STAMM-WILBRANDT, H.; DUECK, G. (2000). Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, Vol.159, 139-171.

163. SERGIENKO, I.V.; SHYLO, V.P.; CHUPOV, S.V.; SHYLO, P.V. (2020). Solving the quadratic assignment problem. *Cybernetics and Systems Analysis*, Vol.56, 53-57.

164. SGHIR I.; BEN JAAFAR I.; GHEDIRA K. (2018). A Multi-Agent Based Optimization Method for Combinatorial Optimization Problems. *International Journal On Artificial Intelligence Tools*, Vol. 27, Article Number: 1850021.

165. SHERALI H.D.; ADAMS, W.P. (1998). *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Dordrecht-Boston-London, Kluwer.

166. SHYLO, P.V. (2017). Solving the quadratic assignment problem by the repeated iterated tabu search method. *Cybernetics and Systems Analysis*, Vol.53, 308-311.

167. SIARRY, P. (ed.) (2016). *Metaheuristics*. Cham: Springer.

168. SILVA A.; COELHO L. C.; DARVISH M. (2021) Quadratic assignment problem variants: A survey and an effective parallel memetic iterated tabu search. *European Journal of Operational Research*, Vol.292, 1066-1084.

169. SIMON, D. (2013). *Evolutionary Optimization Algorithms. Biologically Inspired and Population-Based Approaches to Computer Intelligence*. Hoboken: Wiley.

170. SIVANANDAM, S.N.; DEEPA, S.N. (2008). *Introduction to Genetic Algorithms*. Heidelberg-New York: Springer.

171. SÖRENSEN, K.; SEVAUX, M.; GLOVER, F. (2018) A history of metaheuristics. In MARTÍ, R.; PARDALOS, P.M.; RESENDE, M.G.C. (eds.), *Handbook of Heuristics*, Cham: Springer International Publishing, 1-18.

172. STEINBERG, L. (1961). The backboard wiring problem: a placement algorithm. *SIAM Review*, Vol.3, 37-50.

173. STÜTZLE, T. (2006). Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, Vol.174, 1519-1539.

174. STÜTZLE, T.; DORIGO, M. (1999). ACO algorithms for the quadratic assignment problem. In CORNE, D.; DORIGO, M.; GLOVER, F. (eds.), *New Ideas in Optimization*, London: McGraw-Hill, 33-50.

175. SUN, J.; ZHANG, Q.; YAO, X. (2014). Meta-Heuristic Combining Prior Online and Offline Information for the Quadratic Assignment Problem. *IEEE Transactions on Cybernetics*, Vol.44, 429-444.

176. SYSWERDA, G. (1989). Uniform crossover in genetic algorithms. Proceedings of the Third International Conference on Genetic Algorithms, SCHAFFER, J.D. (ed.), San Mateo: Morgan Kaufmann, 2-9.

177. SZWED P.; CHMIEL W.; KADŁUCZKA P. (2015). OpenCL implementation of PSO algorithm for the quadratic assignment problem. In: Rutkowski L.; Korytkowski M.; Scherer R.; Tadeusiewicz R.; Zadeh L.; Zurada J. (eds.), *Artificial Intelligence and Soft Computing, ICAISC 2015, Lecture Notes in Computer Science*, Vol.9120, Cham: Springer, 223-234.

178. TAILLARD, E. (1991). Robust taboo search for the QAP. *Parallel Computing*, Vol.17, 443-455.

179. TAILLARD, E. (1995). Comparison of iterative searches for the quadratic assignment problem. *Location Science*, Vol.3, 87-105.

180. TALBI, E.G. (2009). *Metaheuristics: From Design to Implementation (Wiley Series on Parallel and Distributed Computing)*. Hoboken: Wiley.

181. TALBI, E.G. (ed.). (2013). *Hybrid Metaheuristics (Studies in Computational Intelligence, Vol.434)*. Heidelberg-New York-Dordrecht-London: Springer.

182. TANG, J.; LIM, M.-H.; ONG, Y.S.; ER, M.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, 1399-1416.

183. TASGETIREN, F.; CHEN, A.; GENCYILMAZ, G.; GATTOUFI, S. (2009). Smallest position value approach. In ONWUBOLU, G.C.; DAVENDRA, D. (eds.), *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization. Studies in Computational Intelligence*, Vol.175, Berlin, Heidelberg: Springer, 121-138.

184. TATE, D.M.; SMITH, A.E. (1995). A genetic approach to the quadratic assignment problem. *Computers & Operations Research*, Vol.22, 73-83.

185. THAINIAM, P. (2019). The effects of memes on memetic algorithms for solving quadratic assignment problem. In Proceedings of 2019 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), IEEE, Piscataway, pp. 1334–1338.

https://doi.org/10.1109/IEEM44572.2019.8978780

186. TOSUN, U. (2014). A new recombination operator for the genetic algorithm solution of the quadratic assignment problem. *Procedia Computer Science*, Vol.32, 29-36.

187. TOSUN, U.; DOKEROGLU, T.; COSAR, A. (2013). A robust island parallel genetic algorithm for the quadratic assignment problem. *International Journal of Production Research*, Vol.51, 4117-4133.

188. UMBARKAR, A.J.; SHETH, P.D. (2015). Crossover operators in genetic algorithms: a review. *ICTACT Journal on Soft Computing*, Vol.6, 1083-1092.

189. VAESSENS, R.J.M.; AARTS, E.H.L.; LENSTRA, J.K. (1998). A local search template. *Computers & Operations Research*, Vol.25, 969-979.

190. VOß, S.; MARTELLO, S.; OSMAN, I.H.; ROUCAIROL, C. (eds.). (1998). *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Boston: Kluwer.

191. WILHELM, M.; WARD, T. (1987). Solving quadratic assignment problems by simulated annealing. *IIE Transactions*, Vol.19, 107-119.

192. YANG, X.-S. (2010). *Engineering Optimization: An Introduction with Metaheuristic Applications*. Hoboken: Wiley.

193. YANG, X.-S. (2010). *Nature-Inspired Metaheuristic Algorithms*. Frome, Luniver Press.

194. ZAMANI, R.; AMIRGHASEMI, M. (2020). A self-adaptive nature-inspired procedure for solving the quadratic assignment problem. In KHOSRAVY, M.; GUPTA, N.; PATEL, N.; SENJYU, T. (eds.). *Frontier Applications of Nature Inspired Computation. Springer Tracts in Nature-Inspired Computing*, Singapore: Springer, 119-147.

195. ZÄPFEL, G.; BRAUNE, R.; BÖGL, M. (2010). *Metaheuristic Search Concepts. A Tutorial with Applications to Production and Logistics*. Berlin-Heidelberg: Springer.

196. ZHOU, Y.; HAO, J.-K.; DUVAL, B. (2020). Frequent pattern-based search: a case study on the quadratic assignment problem. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 1–13. https://doi.org/10.1109/TSMC.2020.3027860

197. ZHU, W.; CURRY, J.; MARQUEZ, A. (2010). SIMD tabu search for the quadratic assignment problem with graphics hardware acceleration. *International Journal of Production Research*, Vol.48, 1035-1047.

198. XU, Y.-L.; LIM, M.H.; ONG, Y.S.; TANG, J. (2006). A GA-ACO-local search hybrid algorithm for solving quadratic assignment problem. In KEIJZER, M. et al. (eds.), *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO 2006* (Siettle, USA), Vol.1, New York: ACM Press, 599-605.

CURRICULUM VITAE

Dovilė Verenė
dovile.kuznecovaite@ktu.lt

Išsilavinimas
2008–2012 m. bakalauro studijos Kauno technologijos universiteto Informatikos fakultete; informatikos bakalauro laipsnis.
2012–2014 m. magistrantūros studijos Kauno technologijos universiteto Informatikos fakultete; informatikos magistro laipsnis.
2014–2019 m. Informatikos doktorantūros studijos Kauno technologijos universiteto Informatikos fakultete.

Profesinė patirtis: Nuo 2013 m. lektorė Kauno technologijos universiteto Informatikos fakultete.

8 AUTORĖS PUBLIKACIJŲ DISERTACIJOS TEMA SĄRAŠAS

STRAIPSNIAI RECENZUOJAMUOSE MOKSLO LEIDINIUOSE

Web of Science duomenų bazėje indeksuotuose leidiniuose su cituojamumo rodikliu

1. [S1; LT] Misevičius, Alfonsas; **Kuznecovaitė, Dovilė**; Platužienė, Jūratė. Some further experiments with crossover operators for genetic algorithms // Informatica. Vilnius : Vilniaus universiteto leidykla. ISSN 0868-4952. eISSN 1822-8844. 2018, vol. 29, iss. 3, p. 499-516. DOI: 10.15388/Informatica.2018.178. [Science Citation Index Expanded (Web of Science); Scopus] [IF: 1,928; AIF: 2,188; IF/AIF: 0,881; Q1 (2018, InCites JCR SCIE)] [M.kr.: N 009] [Indėlis: 0,333]

2. [S1; CH] Misevičius, Alfonsas; **Verenė, Dovilė**. A hybrid genetic-hierarchical algorithm for the quadratic assignment problem // Entropy. Basel : MDPI. ISSN 1099-4300. 2021, vol. 23, iss. 1, art. no. 108, p. 1-31. DOI: 10.3390/e23010108. [Science Citation Index Expanded (Web of Science); Scopus; DOAJ] [IF: 2,494; AIF: 3,264; IF/AIF: 0,764; Q2 (2019, InCites JCR SCIE)] [M.kr.: N 009] [Indėlis: 0,500]

Kituose recenzuojamuose mokslo leidiniuose

1. [S4; LT] Misevičius, Alfonsas; **Kuznecovaitė (Verenė), Dovilė**. Dvieju lygiu iteracinis tabu paieškos algoritmas kvadratinio paskirstymo uždaviniui = A 2-level iterated tabu search algorithm for the quadratic assignment problem // Informacijos mokslai. Vilnius : Vilniaus universitetas. ISSN 1392-0561. eISSN 1392-1487. 2019, t. 85, p. 530-546. DOI: 10.15388/IM.2019.85.19. [Scopus] [M.kr.: N 009] [Indėlis: 0,500]

2. [S4; LT] Misevičius, Alfonsas; **Kuznecovaitė, Dovilė**. Investigating some strategies for construction of initial populations in genetic algorithms // Computational Science and Techniques [elektroninis išteklius]. Klaipėda : Klaipėda University. eISSN 2029-9966. 2017, Vol.5, iss. 1, p. 560-573. DOI: 10.15181/csat.v5i1.1277. [Index Copernicus (IC)] [M.kr.: 09P] [Indėlis: 0,500]

Konferencijų pranešimų medžiagoje

1. [P1e; LT] Stanevičienė, Evelina; Misevičius, Alfonsas; **Kuznecovaitė, Dovilė**; Drąsutė, Vida. Hybrid genetic algorithms for image processing: a short review = Hibridinių genetinių algoritmų taikymas vaizdų apdorojimo uždaviniams spręsti: trumpa apžvalga // ALTA'15 : Pažangios mokymosi technologijos : konferencijos pranešimų medžiaga, 2015 m. gegužės 7 d. = Advanced learning technologies : conference proceedings, 7th of May 2015 / Nacionalinė distancinio mokymo asociacija, Kauno technologijos universitetas, Nacionalinė skaitmeninė koalicija, Baltijos edukacinių technologijų institutas. Kaunas : Kauno technologijos universitetas. ISSN 2335-2140. 2015, p. 45-49. [M.kr.: 07S] [Indėlis: 0,250]

MOKSLINIŲ TYRIMŲ REZULTATŲ SKELBIMAS KONFERENCIJOSE

Konferencijų tezės ir pranešimai

1. [T2; LT] Misevičius, Alfonsas; **Kuznecovaitė, Dovilė;** Platužienė, Jūratė. Eksperimentai su genetiniams algoritmams skirtomis kryžminimo procedūromis // Kompiuterininkų dienos - 2017. Vilnius : Lietuvos kompiuterininkų sąjunga, 2017. ISBN 9789986343264. p. 38. [M.kr.: 07T]

2. [T2; LT] Misevičius, Alfonsas; **Kuznecovaitė, Dovilė.** Dviejų lygių iteracinis tabu paieškos algoritmas kvadratinio paskirstymo uždaviniui // Kompiuterininkų dienos - 2019. Vilnius : Lietuvos kompiuterininkų sąjunga, 2017. ISBN 9789986343264. p. 38. [M.kr.: 07T]

9 PRIEDAI

9.1 Naudotų algoritmų (procedūrų) sudėtingumas

Kryžminimo procedūros:

vieno taško kryžminimo procedūra – $O(n^2)$, n – uždavinio (įėjimo duomenų) apimtis (dydis);

dviejų taškų kryžminimo procedūra – $O(n^2)$;

tolygiojo pasiskirstymo kryžminimo procedūra – $O(n^2)$;

sumaišymo kryžminimo procedūra – $O(n^2)$;

dalinio atvaizdavimo kryžminimo procedūra – $O(n^2)$;

sukeitimų kryžminimo procedūra – $O(n^2)$;

euristinė sukeitimų kryžminimo procedūra – $O(n^2)$;

modifikuota euristinė sukeitimų kryžminimo – $O(n^2)$;

ciklinio kryžminimo procedūra – $O(n^2)$;

suliejimo kryžminimo procedūra – $O(n^3)$;

universaliojo kryžminimo procedūra – $O(n^2)$;

daugelio tėvų kryžminimo procedūra – $O(PD \times n^2)$, PD – populiacijos dydis.

Mutavimo procedūros:

atsitiktinių porinių elementų sukeitimų mutavimas – $O(\xi \times n^2)$, ξ – mutavimo stiprumas;

atsitiktinio rakto mutavimas – $O(\xi \times n^2)$;

priešinių reikšmių mutavimas – $O(\xi \times n^2)$;

maksimalaus atstumo mutavimas – $O(\xi \times n^2)$;

modifikuotas atsitiktinių sukeitimų mutavimas (I) – $O(\xi \times n^2)$;

modifikuotas atsitiktinių sukeitimų mutavimas (II) – $O(\xi \times n^2)$;

kombinuotas mutavimas – $O(\xi \times n^2)$;

godžios adaptyvios paieškos mutavimas – $O(\xi \times n^2)$;

godžios randomizuotos adaptyvios paieškos mutavimas – $O(\xi \times n^2)$;

randomizuotos lokalios paieškos mutavimas – $O(\xi \times n^2)$;

randomizuotos dalinės lokalios paieškos mutavimas – $O(\xi \times n^2)$;

išsklaidytas mutavimas – $O(\xi \times n^2)$.

Tikslo funkcijos (TF) apskaičiavimas – $O(n^2)$;

TF pokyčio apskaičiavimas, kai nenaudojama TF pokyčių atmintis – $O(n)$;

TF pokyčio apskaičiavimas, kai naudojama TF pokyčių atmintis – $O(1)$;

Tabu paieškos (TP) algoritmas – $O(\tau \times n^2)$, τ – TP iteracijų skaičius (nepriklausantis nuo uždavinio dydžio n);

Iteratyviosios tabu paieškos (ITP) algoritmas – $O(Q \times \tau \times n^2)$, Q – ITP iteracijų skaičius;

Hierarchinės iteratyviosios tabu paieškos (HITP) algoritmas – $O(Q^* \times \tau \times n^2)$, $Q^* = (Q \times Q_1 \times Q_2 \times Q_3 \times Q_4 \times Q_5 \times Q_6 \times Q_7)$, $Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7$ žymi atskirų hierarchinių lygių iteracijų skaičius;

Genetinis-hierarchinis algoritmas – $O(PD \times Q^* \times \tau \times n^2 + N_{gen} \times Q^* \times \tau \times n^2)$, PD – populiacijos dydis, N_{gen} – GH algoritmo generacijų skaičius.

UDK 004.421.2(043.3)

SL 344. 2022-*-*, * leidyb. apsk. l. Tiražas * egz. Užsakymas * .
Išleido Kauno technologijos universitetas, K. Donelaičio g. 73, 44249 Kaunas
Spausdino leidyklos „Technologija“ spaustuvė, Studentų g. 54, 51424 Kaunas