KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PRAKTINES INFORMATIKOS KATEDRA

Andrius Milinis

Hibridinis genetinis algoritmas ir jo modifikacijos kvadratinio pasiskirstymo uždaviniui spresti

Vadovas: doc. dr. A. Misevicius

Kaunas, 2005

Turinys

1. Pratarme

Siekis išspresti sudetingus optimizavimo uždavinius, paskatino euristiniu optimizavimo algoritmu, tokiu kaip atkaitinimo modeliavimas (angl. *simulated annealing*)[3], skruzdeliu koloniju elgsenos imitavimo (angl. *ant colony optimization*)[9], tabu paieška (angl. *tabu search*) [15,16], godžioji randomizuota adaptyvioji paieškos proceduros (angl. *greedy randomized adaptive search procedures* (*GRASP*)) [28] ir kitokiu algoritmu atsiradima. Kuriant greitai veikiancius euristinius algoritmus, remiamasi procesais, kurie vyksta mus supancioje aplinkoje. Kombinatorinio optimizavimo uždaviniams spresti vieni iš placiausiai taikomu euristiniu algoritmu yra genetiniai algoritmai (GA). Svarbiausia genetiniu algoritmu ypatybe yra ta, jog jie paremti naturalios atrankos biologiniais procesais, vykstanciais gamtoje. Šiame darbe nagrinejamas praktinis GA panaudojimas, sprendžiant kvadratinio pasiskirstymo (KP) uždavini. Tai gerai žinomas kombinatorinis optimizavimo uždavinys, kuris turi platu praktini pritaikyma.

Terminas „genetinis algoritmas“ nereiškia kokio nors konkretaus algoritmo. Greiciau šis terminas nusako tam tikra, gana universalu meta-metoda – algoritmu su panašiomis savybemis klase. Viena iš tokiu savybiu yra turimu sprendiniu kryžminimas. Ši savybe siejama su procedura, kuri vadinama „krosoveris“.

Sprendiniu kryžminimo procedura atlieka labai svarbu vaidmeni, kuriant efektyvius genetinius algoritmus. Šiame darbe nagrinejamos konceptualios savybes ir specifines charakteristikos ivairiu „krosoverio“ proceduru modifikaciju, sprendžiant kvadratinio pasiskirstymo uždavini. Algoritmai išbandomi panaudojant KP uždavinio testinius pavyzdžius iš testiniu pavyzdžiu bibliotekos – QAPLIB [6].

Darbas pradedamas ivadu, kuriame suformuluojamas kvadratinio pasiskirstymo uždavinys ir aptariama viena iš KP uždavinio taikymo sriciu – elektronines aparaturos projektavimas. Euristiniai optimizavimo metodai, orientuoti šiam uždaviniui, apžvelgti 3 sk. Genetinio algoritmo šablonas ir sprendiniu kryžminimo algoritmo modifikacijos aprašomos 4 sk. 5 sk. pateikiami eksperimentiniu tyrimu rezultatai bei ju analize. Darbas baigiamas išvadomis.

2. Ivadas

2.1. Kvadratinio pasiskirstymo uždavinys: bazines savokos

Kvadratinio paskirstymo (KP) uždavinys (angl. *quadratic assignment problem*) formuluojamas taip [20]: duotos matricos $\boldsymbol{A}=(a_{ij})_{n\times n}$ ir $\boldsymbol{B}=(b_{kl})_{n\times n}$ bei aibe Π, kuria sudaro visi galimi naturiniu skaiciu 1, 2, …, *n* perstatymai. Reikia tarp visu perstatymu iš Π surasti toki perstatyma $p=(p(1), p(2), ..., p(n))$, kuriam esant butu minimizuota funkcija

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

Jeigu matricos ***A*** ir ***B*** yra simetrines, tai gaunamas simetrinio kvadratinio paskirstymo uždavinio atvejis.

Kvadratinio paskirstymo uždavinys pasižymi ivairiais praktiniais taikymais, tarp kuriu paminetini šie: pastatu išdestymas, planavimas [11]; irengimu išdestymas; vaizdu (pilku atspalviu) formavimas; elektronines aparaturos projektavimas (paneliu, klaviaturu projektavimas; elektroniniu komponentu išdestymas spausdinto montažo plokštese ar dideles integracijos mikroschemose). Elektronines aparaturos projektavimas komponentu išdestymo kontekste placiau aptariamas 2 sk.

Kvadratinio paskirstymo uždavinys priklauso NP-sunkiu kombinatorinio optimizavimo uždaviniu klasei [29]. Tokie uždaviniai tiksliai išsprendžiami tik esant labai nedidelems ju apimtims ($n\leq30$). Todel vidutines ir dideles apimties KP uždaviniams spresti naudojami Euristiniai algoritmai. Reikšminga tokiu algoritmu grupe sudaro atkaitinimo modeliavimo (angl. *simulated annealing*) [3], lokaliosios paiškos [26], tabu paieškos (angl. *tabu search*) [30] ir genetiniai (angl. *genetic algorithms*) [1], [12], [23], [31] algoritmai. Jie apžvelgiami 3 sk.

Kadangi kvadratinio paskirstymo uždavinys yra vienas iš kombinatorinio optimizavimo uždaviniu, trumpai apibrešime pagrindines savokas, susijusias su tokiais uždaviniais.

Tarkime, kad S yra kombinatorinio (diskretinio) optimizavimo uždavinio sprendiniu aibe, o $f: S\rightarrow R^1$ – tikslo funkcija. Be to, yra duota aplinkiniu sprendiniu (aplinkos) funkcija N: $S\rightarrow 2^S$, bet kuriam sprendiniui $s\in S$ apibrežianti aibe $N(s)\subseteq S$ − sprendinio s aplinkiniu sprendiniu aibe. Bet kuris sprendinys $s'\in N(s)$ gali buti betarpiškai pasiektas iš sprendinio s, atlikus elementaria sprendinio s pertvarkymo operacija, vadinama perturbacija (perejimu). Paprastai prieš darant perturbacija apskaiciuojama tikslo funkcijos f reikšme; sakoma, jog atliekamas bandymas. Tikslas yra surasti toki sprendini $s^{\bullet} \in S$, kuris minimizuotu funkcija f, t. y. $s^{\bullet} = \arg\min_{s \in S} f(s)$.

Labai dažnai kvadratinio paskirstymo uždaviniui, kaip ir kitiems kombinatorinio optimizavimo uždaviniams, naudojama vadinamoji poriniu sukeitimu funkcija, kitaip tariant, aplinka (žymima N_2). KP uždavinio atveju ši funkcija apibrežiama taip: $?_2(p) = \{p' \mid p' \in \Pi,\ d(p, p') = 2\}$, cia p – bet kuris sprendinys (perstatymas) iš Π, $d(p, p')$ – atstumas tarp sprendiniu p ir p', kuris gali buti išreikštas tokia formule $d(p, p') = \sum_{i=1}^{n} \operatorname{sgn} |p(i) - p'(i)|$ (n yra uždavinio apimtis). Taigi funkcija N_2 apibrežia aibe, sudaryta iš tokiu sprendiniu-kaimynu, kurie gaunami, duotame sprendinyje atlikus 2-ju nariu sukeitima vietomis. Šiuo atveju priimta sakyti, kad atliekamas perejimas iš esamo sprendinio i sprendini-kaimyna. Formaliai perejimas aprašomas panaudojant specialu operatoriu – perturbacija, kurios metu sukeiciama vietomis i-tasis ir j-tasis sprendinio nariai (žymima p_{ij} ($1 \le i, j \le n$, $i \ne j$)). (Tuomet perejimas iš sprendinio p i sprendini p' galetu buti užrašytas, pvz., taip: $p' = p \oplus p_{ij}$.)

Dažniausiai kaimyniniai sprendiniai nagrinejami ne atsitiktinai („aklai“), o pagal tam tikra griežta, fiksuota tvarka. Perturbaciju p_{ij} indeksai i, j, apibrežiantys sprendiniu nagrinejimo eiliškuma, gali buti imami iš masyvo, kuriame indeksai kuriuo nors budu „sumaišomi“ (atskiru atveju išdestomi tvarkingai, pvz., didejimo tvarka); svarbu tik, kad kiekvienas indeksas pasitaikytu po viena karta. Tuomet bendru atveju sprendiniu nagrinejimo tvarka galima aprašyti sudarant tokia seka $\{p_{\mathrm{ISM}(i^{(k)})\mathrm{ISM}(j^{(k)})}\}_{k=1,2,\ldots}$, cia ISM – indeksu „sumaišymo“ masyvas; indeksai $i^{(k)}$, $j^{(k)}$ apskaiciuojami pagal formule

$$\left[\begin{array}{l} i^{(k)} = \mathrm{if}(j^{(k-1)} < n, i^{(k-1)}, \mathrm{if}(i^{(k-1)} < n-1, i^{(k-1)}+1, 1)) \\ j^{(k)} = \mathrm{if}(j^{(k-1)} < n, j^{(k-1)}+1, i^{(k)}+1) \end{array}\right.$$

, kur k – bandymo eiles numeris, $i^{(k)}$, $j^{(k)}$ – nauji indeksai, $i^{(k-1)}$, $j^{(k-1)}$ – buve prieš tai indeksai ($i^{(0)}=1$, $j^{(0)}=1$). (Funkcija if aprašoma taip:

$$\mathrm{if}(\mathrm{salyga}, x_1, x_2) = \begin{cases} x_1, \mathrm{salyga} = \mathrm{TRUE} \\ x_2, \mathrm{salyga} = \mathrm{FALSE} \end{cases}.)$$

Pastebesime, jog KP uždavinio atveju tikslo funkcijos pokytis $\Delta z(p, i, j)$ ($1 \le i, j \le n$, $i \ne j$), gautas, sprendinyje p sukeitus vietomis i-taji ir j-taji narius, t. y., atlikus perturbacija p_{ij}, apskaiciuojamas, atlikus $O(n)$ operaciju:

$$? z(p, i, j) = (a_{ij} - a_{ji})(b_{p(j)p(i)} - b_{p(i)p(j)}) + \sum_{k=1, k \ne i, j}^{n} \left[(a_{ik} - a_{jk})(b_{p(j)p(k)} - b_{p(i)p(k)}) + (a_{ki} - a_{kj})(b_{p(k)p(j)} - b_{p(k)p(i)}) \right] \quad (2)$$

jeigu a_{ii} (arba b_{ii})=const, i=1, 2, ..., n. (O tai reiškia, kad norint išnagrineti visus aplinkos N_2 sprendinius, reikia atlikti $O(n3)$ operaciju.)

Jeigu matricos $\boldsymbol{A}$ ir/arba $\boldsymbol{B}$ yra simetriškos, tai pateiktoji formule žymiai supaprasteja. Pavyzdžiui, jeigu tik vienintele matrica $\boldsymbol{B}$ yra simetriška, o matrica $\boldsymbol{A}$ – ne, tai galima transformuoti nesimetriška matrica $\boldsymbol{A}$ i simetriška matrica $\boldsymbol{A}$¢, sudedant atitinkamus matricos $\boldsymbol{A}$ viršutinio ir apatinio „trikampio" elementus. Tuomet gaunama tokia kompaktiška formule tikslo funkcijos pokyciui apskaiciuoti:

$$? z(p, i, j) = \sum_{k=1, k \ne i, j}^{n} (a'_{ik} - a'_{jk})(b_{p(j)p(k)} - b_{p(i)p(k)}), \quad (3)$$

kur $a'_{ik} = a_{ik} + a_{ki}$, $\forall i, k \in \{1, 2, ..., n\}$, $i \ne k$.

2.2. Uždavinio pritaikymas

Vienas iš aktualiu kvadratinio paskirstymo uždavinio taikymu yra elektronines aparaturos projektavimas. Elektronines aparaturos, tame tarpe integraliniu mikroschemu, automatizuotojo projektavimo procesas apima daug projektavimo etapu [29], iš kuriu pagrindiniai yra:

1. loginis-funkcinis projektavimas;
2. schemotechninis projektavimas;
3. topologijos projektavimas.

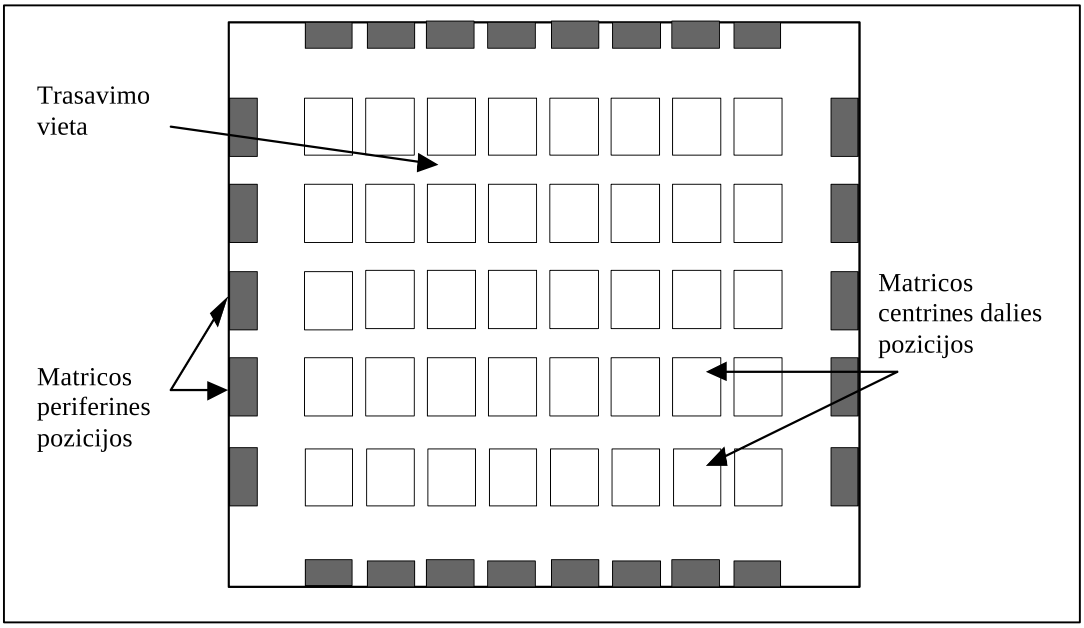
Vienas iš sudetingiausiu yra topologijos projektavimo etapas, apimantis funkciniu mazgu – elementu - komponavimo, išdestymo ir sujungimu trasavimo bei kontroles stadijas. Kuriant automatizuotojo projektavimo sistemas, topologijos projektavimo etapas išskaidomas i dvi santykinai savarankiškas stadijas:

1. Elementu išdestymas;
2. Sujungimu trasavimas.

Tai daroma, siekiant maksimaliai palengvinti labai sudetingo kompleksinio uždavinio sprendima. Sujungimu trasavimo, taigi ir viso topologijos projektavimo etapo, atlikimo kokybe tiesiogiai priklauso nuo išdestymo kokybes. Todel elementu išdestymo uždavinys yra ypatingai aktualus topologijos projektavime, galima sakyti, nulemiantis galutinius rezultatus ir projektavimo sekme.

Elektronineje aparaturoje placiai naudojamas dideles integracijos mikroschemas sudaro tukstanciai topologiniu vienetu (tranzistoriu, rezistoriu ir pan.), kurie apjungiami i stambesnius mazgus - funkcinius elementus (logines schemas, atminties lasteles ir pan.). Funkciniai elementai (arba tiesiog elementai) - tai tam tikri nedalomi vienetai, kuriais operuojama išdestymo uždavinyje.

Dideles integracijos mikroschemos dažnai realizuojamos vadinamuose baziniuose matriciniuose kristaluose (BMK). Elementai destomi jiems skirtose matricinio kristalo pozicijose. Reguliarios strukturos kristaluose šiu poziciju padetys yra griežtai apibrežtos: pozicijos išdestomos horizontaliose eilutese ir vertikaliuose stulpeliuose. Poziciju gabaritai (plotis, aukštis) ir forma yra vienodi. Matricinio kristalo konstrukcijoje paprastai išskiriamos centrines dalies pozicijos bei periferines pozicijos (žr. 1 pav.).



1 pav. Matricinio kristalo struktura

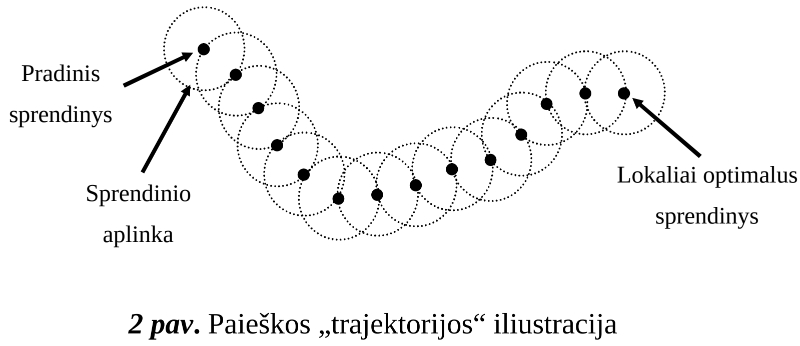
Aprašant komponentu išdestymo uždavini kaip kvadratinio paskirstymo uždavini, matrica $\boldsymbol{A}=(a_{ij})$ interpretuojama kaip elektriniu ryšiu tarp komponentu matrica, kur a_{ij} yra ryšiu („grandiniu”), jungianciu i-taji komponenta su j-tuoju komponentu, skaicius. Matrica

$\boldsymbol{B}=(b_{kl})$ yra atstumu matrica, kur b_{kl} – atstumas (staciakampeje arba Euklido metrikoje) tarp k-tosios ir l-tosios destymo pozicijos. Perstatymas $p=(p(1), p(2), ..., p(n))$ apibrežia komponentu išdestyma i pozicijas, t. y., išdestymo konfiguracija (cia $p(i)$ – pozicijos, i kuria paskirtas i-tasis komponentas, numeris). Tuomet $z(p)$ yra bendras ryšiu (sujungimu) tarp komponentu ilgis, kuri reikia minimizuoti tam, kad sudaryti kuo palankesnes salygas tolimesniems projektavimo etapams.

3. Metodu KP uždaviniui spresti apžvalga

3.1. Klasikiniai lokaliai optimaliu sprendiniu paieškos algoritmai

Iš pradžiu aptarkime klasikinio lokaliai optimaliu sprendiniu paieškos algoritmo atveji, grindžiama vadinamaja „nusileidimo“ („godžiosios“ paieškos) metodologija [26]. „Nusileidimo“ algoritmai yra santykinai labai paprasti, bet pakankamai efektyvus; jie laikytini sudetingesniu euristiniu algoritmu ir metodu ištakomis. Išskiriamos šios dvi svarbios „nusileidimo“ strategijos: a) „greiciausias nusileidimas“; b) „staigiausias nusileidimas“. Abiem atvejais paieška pradedama nuo kurio nors pradinio sprendinio. Toliau paieškos procesas vykdomas nuosekliai atliekant sprendiniu transformacijas ir „pereinant“ nuo vieno sprendinio s prie kito s' iš esamo sprendinio aplinkos $N(s)$. Abieju tipu „nusileidimo“ algoritmuose sprendimas, ar pakeisti esama sprendini s nauju s' (t.y., ar „pereiti“ prie naujo sprendinio iš esamo aplinkos), ar ne, yra determinuotas. Jis yra teigiamas tik tada, kai naujasis sprendinys yra geresnis už esamaji (kas minimizavimo uždavinio atveju reiškia, jog TF reikšmiu, apskaiciuotu naujam ir esamam sprendiniui, pokytis $\Delta f = f(s') - f(s)$ yra neigiamas ($\Delta f < 0$)). Skirtumas tarp strategiju yra tik tas, kad pirmuoju atveju „pereinama“ prie pirmojo „sutikto“ geresnio sprendinio iš esamo aplinkos, o antruoju atveju — prie geriausio sprendinio iš esamo aplinkos. Del to pirmoji strategija yra vadinama „pirmojo pagerejimo“ strategija, o antroji — „geriausio pagerejimo“. Paieškos procedura (procesas) tesiama tol, kol esamas sprendinys s nera lokaliai optimalus, t.y. sprendinio s aplinkoje yra bent vienas geresnis sprendinys (žr. 2 pav.).



2 pav. Paieškos „trajektorijos“ iliustracija

Klasikinio („nusileidimo“) algoritmo šablonas pateikta 3 pav.

```
procedure klasikinis_lokaliosios_paieškos_algoritmas;
//pradiniai duomenys: s° — pradinis sprendinys; //
//rezultatai: s — rastas lokaliai optimalus sprendinys //
begin
  s:=s°;
  while s nera lokaliai optimalus do begin
    //vykdomas „godžiosios paieškos“ ciklas //
    parinkti sprendini s′ iš esamo sprendinio s aplinkos N(s),
    ir toki, kad: f(s′) – f(s)<0;
    s:=s′ //esamas sprendinys s pakeiciamas nauju sprendiniu s′ //
          //(„pereinama“ prie naujo sprendinio) //
  end //while //
end.
```

3 pav. Klasikinio lokaliai optimaliu sprendiniu paieškos algoritmo šablonas

Akivaizdus klasikiniu euristiniu algoritmu trukumas yra paieškos ribotumas; t.y. apsiribojama tik vieno lokaliai optimalaus sprendinio suradimu. Savaime suprantama, jog toks surastas sprendinys, nors ir yra lokaliai optimizuotas, gali visai nebuti geros kokybes. Dar daugiau, galima netgi prognozuoti, kad toks sprendinys tiketina kaip tik bus nepakankamos kokybes — juk tai tik vienas galimas lokaliai optimalus sprendinys iš didžiulio lokaliuju optimumu (LO) skaiciaus. Naturalu, kad siekiama sukurti tokius euristinius algoritmus (EA), kurie leistu iveikti mineta esmini klasikiniu EA trukuma, t.y. igalintu bandyti išvengti „ikritimu“ i lokaliuju optimumu „duobes“ be galimybiu iš ju „išeiti“. Naujai kuriamu, vadinamuju moderniuju euristiniu (ME) (arba meta-euristiniu) metodu vienas iš tikslu ir buvo jei ne išvengti, tai bent minimizuoti „lokaliuju optimumu duobiu“ fenomeno neigiamas pasekmes. Supaprastintas (vienas iš galimu) moderniuju euristiniu metodu apibudinimas galetu buti toks: modernioji euristika yra uždavinio sprendimo metodas, orientuotas keliu (daugiau nei vieno) lokaliai optimaliu sprendiniu paieškai. Taigi ME algoritmu atveju atsiranda didesnes potencialios galimybes surasti aukštesnes kokybes sprendinius, nes galima mastyti taip: didejant surandamu LO skaiciui, dideja ir tikimybe, jog kuris nors surastas sprendinys bus geresnis, negu pirmasis surastas lokaliai optimalus sprendinys. Aišku, paieškos laikas atitinkamai pailgeja, taciau „racionalusis grudas“ yra tame, kad yra geriau gauti aukštos kokybes rezultata per ilgesni laika, negu tenkintis blogu rezultatu, gautu per labai trumpa laika. Esme gludi principe: „geresni rezultatai — per ilgesni (bet priimtina) laika“. Net ir pakankamai ilgas (bet tenkinantis vartotoja ar tyrinetoja) paieškos laikas yra pateisinamas, jeigu tik tai suteikia galimybes apciuopiamai pagerinti sprendiniu kokybe. Žinoma, nuolatos turi buti stengiamasi, kad ir aukštos kokybes, kuo arciau globaliojo optimumo esantys sprendiniai butu pasiekiami per kuo trumpesni laika — tai išliks pagrindiniu siekiu tiems, kurie tobulina, modifikuoja esamus ar kuria naujus euristinius algoritmus.

3.2. Atkaitinimo modeliavimas

Atkaitinimo modeliavimo (AM) metodo ištakos slypi statistineje mechanikoje, o jeigu tiksliau, energetiniu procesu, vykstanciu sistemose, sudarytose iš didelio skaiciaus daleliu, imitavime. Šio imitavimo esme slypi tame, jog iš pradžiu sistema „pervedama“ i dideles energijos busena, o po to, palaipsniui mažinant energija, stengiamasi pasiekti busena, atitinkancia žemiausia sistemos energetini lygmeni – tarsi kunas butu ikaitintas iki pakankamai aukštos temperaturos, o paskui, ji atkaitinant, t.y. mažinant temperatura, jis butu savotiškai „užgrudinamas“. Šio modeliavimo pradininkai (1953 m.) buvo Metropolis, Rozenblutas ir kt. Jie pasinaudojo Bolcmano (eksponentiniu) pasiskirstymo desniu apibreždami tikimybe, kad sistema, ivykus joje tam tikrai perturbacijai, pereis iš vieno energijos lygio (E_1) i kita (E_2), kai temperatura lygi t: $\mathrm{P} = \begin{cases} 1, \Delta E < 0, \\ \mathrm{e}^{-\Delta E / C_B t}, \Delta E \geq 0 \end{cases}$, cia $\Delta E = E_2 - E_1$, o C_B – Bolcmano konstanta. Cerný (1982 m.) [2] ir Kirkpatrikas su bendraautoriais (1983 m.) [5] buvo pirmieji, kurie panaudojo atkaitinimo modeliavimo metoda sprendžiant kombinatorinio optimizavimo uždavinius.

Konkrecios atkaitinimo modeliavimo algoritmu realizacijos skiriasi viena nuo kitos šiais veiksniais (faktoriais): aplinkos funkcija, atkaitinimo („atšaldymo“) schema ir baigimo salyga. Viena iš dažniausiai naudojamu aplinkos funkciju yra vadinamoji poriniu sukeitimu funkcija N_2. Yra dvi galimos alternatyvos peržiurint „sprendinius-kaimynus“: pirma, pasirinkti einamojo sprendinio „kaimyna“ atsitiktinai; antra, nagrineti einamojo sprendinio aplinka pagal tam tikra fiksuota, determinuota tvarka, pvz., nuosekliai vykdant perturbacijas p_{ij}, kai i kinta nuo 1 iki n–1, o j – nuo i+1 iki n (norint išnagrineti visus aplinkos N_2 sprendinius, reikia atlikti $O(n^3)$ operaciju).

Atkaitinimo modeliavimo algoritmuose naudojamas vadinamasis ekvilibriumo testas tam, kad nustatyti, kuriuo butent momentu turi buti mažinama temperatura. Ekvilibriumas trumpai gali buti charakterizuojamas kaip tam tikra stabili, be žymesniu fliuktuaciju proceso busena. Kombinatoriniu uždaviniu atveju ekvilibriumo kriterijumi galetu buti, pvz., tikslo funkcijos reikšmiu svyravimu amplitude (jeigu ji nedidele, tai konstatuojama, jog ekvilibriumas pasiektas). Atsižvelgiant i tai, kaip realizuojamas ekvilibriumo testas, skiriami du atkaitinimo tipai: homogeninis ir nehomogeninis atkaitinimas. Pirmuoju atveju atliekama daug bandymu esant tai paciai, fiksuotai temperaturai; tai tesiama tol, kol pasiekiamas ekvilibriumas – tada temperatura sumažinama, ir procesas kartojamas. Antruoju atveju temperatura mažinama po kiekvieno ivykdyto bandymo; galima sakyti, jog ekvilibriumo testas iš viso neatliekamas.

Modeliuojant atkaitinima svarbu parinkti tinkama temperaturos mažinimo formule. Praktiškai taikomuose atkaitinimo algoritmuose placiausiai naudojamos yra: geometrine formule ($t_k = a \cdot t_{k-1}$; tk – einamoji temperaturos reikšme; k=1, 2, ...; t_0 = const; 0.8 ≤ a ≤ 0.99) ir Lundy-Mees formule ($t_k = t_{k-1} / (1 + b t_{k-1})$; k=1, 2, ...; t_0 = const; $b \ll t_0$).

Tiesa, naujausiose atkaitinimo modeliavimo algoritmu versijose temperatura greiciau keiciama periodiškai nei monotoniškai mažinama. Kai kuriu tyrimu rezultatai liudija, kad vietoj tiesioginio atkaitinimo tikslingiau taikyti tam tikra „atkaitinimu“ ir „kaitinimu“ seka, t. y., vadinamaji reatkaitinima (angl. *reannealing*), kuris suteikia papildomas, lankstesnes galimybes atkaitinimo modeliavimo algoritmams.

Baigimo salyga. Atkaitinimo procesas teoriškai turetu buti tesiamas tol, kol galutine temperatura t_g tampa lygi 0. Taciau praktiškai atkaitinima galima baigti ir anksciau – kai tampa mažai tiketina, jog bus pasiektas pagerinimas. Dažnai kaip baigimo salyga tarnauja fiksuotas atliekamu bandymu skaicius.

3.3. Tabu paieška

Tabu paieškos (TP) metodas buvo pasiulytas Gloverio [15], [16]. Šis metodas remiasi išplesta lokaliaja paieška. Skirtingai, negu iprasti klasikiniai algoritmai, kurie apsiriboja lokaliai optimalaus sprendinio suradimu nagrinejamo sprendinio aplinkoje, TP pagristi algoritmai tesia paieška ir tuo atveju, kai surandamas lokaliai optimalus sprendinys, t.y. duoto sprendinio aplinkoje neimanoma surasti geresnio sprendinio. Tabu paieška yra iteracinis geresniu sprendiniu paieškos procesas, igalinantis daug kartu „pereiti“ nuo vieno lokaliojo optimumo (LO) prie kito isimenant geriausiaji iš surastu sprendiniu.

Pagrindine TP ideja yra leidimas atlikti „perejimus“ net ir tais atvejais, kai naujasis sprendinys iš duotojo sprendinio aplinkos nera geresnis už duotaji sprendini. Taciau kai kurie „perejimai“ turi buti draudžiami, t.y. tabu paieška yra pagrista draudimu metodologija: draudimai yra butini tam, kad neleisti grižti i tuos pacius, jau nagrinetus sprendinius ir tokiu budu išvengti pasikartojimu (ciklinimu).

Paieškos procesas pradedamas nuo pradinio, galbut atsitiktinai sugeneruoto, sprendinio s iš aibes S, po to „perejimai“ iš vieno sprendinio i kita vykdomi iteraciniu budu. Paieškos eigoje analizuojama einamojo sprendinio s aplinkos sprendiniu aibe $N(s)$, ir „pereinama“ i ta sprendini $s' \in N(s)$, kuriam tikslo funkcijos f reikšme yra mažiausia. Taciau „perejimas“ gali buti atliekamas ir tuo atveju, kai tikslo funkcijos (TF) pokytis yra teigiamas, t.y. „perejimas“ pablogina einamaja tikslo funkcijos reikšme. Trumpai tariant, pasirenkamas tas „perejimas“, kuris mažiausiai pablogina TF reikšme – taip galima „pereiti“ nuo vieno lokaliai optimalaus sprendinio prie kito. Grižimas i anksciau „aplankyta“ sprendini turi buti uždraudžiamas tam tikram laikotarpiui – kad butu išvengta „ciklinimosi“, t.y. paieškos kartojimo iš naujo nuo to pacio „taško“. Tokiu budu nagrinetieji sprendiniai tam tikrais momentais tampa „tabu“: jie (arba atitinkamu „perejimu pedsakai“) itraukiami i specialu saraša – vadinamaji tabu saraša (atminti) T. Taigi „perejimas“ i sprendini $s' \in N(s)$ yra draudžiamas, traktuojamas kaip tabu, jeigu tas sprendinys arba atitinkamas „perejimas“ duotu metu yra saraše T. Dydis $h=|T|$ (tabu sarašo ilgis) yra labai svarbus: jeigu jis yra labai mažas, tai gali buti gaunamas nepageidaujamas ciklinimasis; jeigu – pernelyg didelis, tai gali buti apribota paieška atskirose sprendiniu „erdves“ srityse.

Tiesmukiškas „perejimu“ draudimas gali sumažinti paieškos efektyvuma. Be to, grižimas i anksciau nagrinetus sprendinius, praejus tam tikram laiko intervalui, gali buti labai naudingas. Del šiu priežasciu, kartu su tabu sarašu naudojamas ir vadinamasis aspiracijos kriterijus. Jo pagalba galima „nekreipti demesio“ i esama „tabu busena“, anuliuoti ja, esant tam tikroms, „palankioms“ aplinkybems. Vienas iš aspiracijos kriteriju yra toks: „perejimas“ iš sprendinio s i sprendini s' – nors jis ir yra tabu – leidžiamas, jeigu tenkinama salyga $f(s')<f(s^*)$, kur s^* yra geriausias iki duotojo momento surastas sprendinys. Tokiu budu tipine TP algoritmuose naudojama sprendimo priemimo taisykle yra tokia: s pakeiciamas s' su salyga, kad: $f(s')<f(s^*)$ arba ($s' = \arg\min_{s'' \in T(s)} f(s'')$ ir s' nera tabu).

Bendroji tabu paieškos algoritmu bazine struktura pateikiama žemiau:

```
procedure tabu_paieška;
// pradiniai duomenys: s — pradinis sprendinys;  rezultatai: s* — geriausias rastas sprendinys //
begin
  s*:=s;
  inicializuoti tabu saraša T;
  repeat  // vykdomas tabu paieškos ciklas //
    rasti geriausia sprendini s′∈N′(s)⊆N(s),
    cia N′(s) – sprendinio s aplinkos poaibis, kurio sprendiniai
    (arba "perejimo" iš s i N(s) „pedsakai") nepriklauso tabu sarašui T
    arba tenkina aspiracijos salyga;
    s:=s′;  // esamas sprendinys pakeiciamas nauju ir naudojamas kaip „išeities taškas“
          // tolimesnese iteracijose //
    isiminti sprendini s (arba "perejimo" iš s i s¢ „pedsaka") tabu
    saraše T;
    if f(s)<f(s*) then s*:=s;  // isimenamas geriausias sprendinys //
    jei reikia, atnaujinti tabu saraša T
  until patenkinta baigimo salyga
end.
```

4 pav. Tabu paieškos šablonas

Tabu paieškos algoritmai skiriasi vienas nuo kito, atsižvegiant i tabu sarašo realizavima, aspiracijos kriterijaus parinkima bei kitus faktorius, tokius kaip papildoma („ilgalaike“) tabu atmintis, paieškos intensifikavimo, diversifikavimo mechanizmai ir kt. TP algoritmu lankstumui padidinti tiek tabu sarašas, tiek aspiracijos kriterijus gali buti dinamiškai atnaujinami paieškos vykdymo eigoje. TP vykdymas paprastai baigiamas, atlikus tam tikra fiksuota paieškos iteraciju skaiciu. TP rezultatas yra geriausias paieškos proceso eigoje „isimintas“ sprendinys.

3.4. Genetiniai algoritmai

Genetiniai algoritmai (GA) ir ju sudarymo principas buvo pasiulytas XX-ojo amžiaus 70-aisiais metais Hollando [22]. Pradedant Hollando pirmuoju darbu, genetiniai algoritmai buvo sekmingai išbandyti, sprendžiant ivairius optimizavimo uždavinius, pvz., tokius, kaip elektroniniu komponentu išdestymas [12], operaciju planavimas ir kt. Genetiniai algoritmai (GA), ju veikimas yra pagristas evoliucijos, vykstancios gyvojoje gamtoje, t. y., naturaliosios atrankos proceso imitavimu. Pagrindines savokos, kurios naudojamos modeliuojant biologines evoliucijos procesus, yra „individas“ ir „populiacija“. „Individas“ yra tam tikras elementarus, daugiau neskaidomas vienetas, objektas. Didesne ar mažesne „individu“ grupe sudaro „populiacija“. Dar vienas svarbus dalykas yra vadinamasis „individo tinkamumas“ – savotiška „individo verte“. „Individo verte“ galima traktuoti kaip „individo“ sugebejimo sekmingai prisitaikyti (prie aplinkos), išlikti ir reprodukuotis laipsni. Šia prasme „vertingesnis“ (grynai biologiškai) yra tas „individas“, kuris sugeba geriausiai prisitaikyti (buti „stipresnis“ už kitus) ir, gal but, palikti didesni palikuoniu („vaiku“) skaiciu.

Optimizavime vietoje savoku „individas“, „populiacija“, „individo verte“ naudojamos tradicines, iprastos savokos: „individui“ atitinka atskiras sprendinys, „populiacijai“ – sprendiniu aibe (grupe, rinkinys), pagaliau, „individo verte“ yra asocijuojama su tikslo funkcijos reikšme duotajam sprendiniui. Taip, kaip gyvosios gamtos evoliucijoje išlieka tik „vertingiausi“ („stipriausi“) „individai“, taip ir optimizavime siekiama gauti kuo „geresni“ sprendini, t. y., sprendini su kuo mažesne (ar didesne) – nelygu, koks optimizavimo uždavinys (minimizavimo ar maksimizavimo) sprendžiamas – tikslo funkcijos reikšme.

Iš tikruju, terminas „genetiniai algoritmai“ reiškia ne koki nors atskira, specifini euristini optimizavimo algortima. Greiciau šis terminas nusako tam tikra bendra, gana universalu metoda, strategija – metaeuristika, t. y., šeima euristiniu algoritmu su panašiais veikimo principais ir savybemis. Taigi genetiniai algoritmai priklauso tai euristiku klasei, kuriai budingos sekancias savybes:

1. Operuojama su viena ar daugiau leistinu sprendiniu „populiaciju“.
2. Atskiri sprendiniai iš vienos ar keliu „populiaciju“ parenkami (panaudojant kuria nors euristine taisykle) tolimesniam nagrinejimui, apdorojimui, teikiant pirmenybe tiems sprendiniams, kuri turi „geresnes“ tikslo funkcijos reikšmes.
3. Naudojamas tam tikras „mechanizmas“, kuris skirtas formuoti naujiems leistiniems sprendiniams, kombinuojant (derinant) anksciau gautus sprendinius ar kurias nors ju savybes, charakteristikas.
4. Esant reikalui, panaudojamas papildomas „mechanizmas“ generuoti naujiems galimiems sprendiniams iš atskiru anksciau gautu sprendiniu (atliekant atsitiktinius tu sprendiniu elementu perstatymus (perturbacijas)).
5. Atlikus (2)–(4) žingsnius, esama populiacija atnaujinama, pašalinant iš jos tam tikrus sprendinius (paprastai, „blogesnius“) bei paliekant joje „geresniuosius“ sprendinius, tame tarpe, naujus suformuotus (sugeneruotus), jeigu jie pakankamai „geri“.

Formalizuojant genetiniu algoritmu aprašyma, aukšciau minetos bendrosios savybes, „mechanizmai“ susiejami su atitinkamomis proceduromis. Taip (2) savybe susiejama su vadinamaja atrinkimo procedura, (3) „mechanizmas“ tradiciškai vadinamas krosoveriu, (4)

„mechanizmas“ – mutavimo procedura („krosoveris“). (5) etape (žingsnyje) atliekamas „populiacijos“ atnaujinimas.

Egzistuoja daugybe ivairiu budu, kaip konkreciai realizuoti „sprendiniu-tevu“ parinkimo, krosoverio, mutavimo bei „populiacijos“ sprendiniu atnaujinimo proceduras [19]. Pavyzdžiui, mes galime operuoti su viena didele sprendiniu „populiacija“, taciau galime tureti ir kelias mažesnes („lygiagrecias“) „populiacijas“ („sub-populiacijas“). Toliau, mes galime pasirinkti „tevus“, atsižvelgiant i ju vertinguma (tinkamuma), t. y., absoliutine tikslo funkcijos reikšme – arba i kokius nors kitus ivertinimus, charakteristikas ar kriterijus. Galima labai ivairiai realizuoti krosoverio bei mutavimo proceduras, be to, galima nuspresti apjungti krosoveri ir mutavima i viena procedura arba tai daryti atskiru, nepriklausomu proceduru pagalba. Pagaliau, mes galime apsispresti pakeisti visus be išimties einamosios „populiacijos“ narius (sprendinius) naujai gautais „palikuonimis“ po kiekvieno algoritmo ciklo iteracijos ivykdymo. Dar reikia pastebeti, kad genetinio algoritmo realizacijos gali priklausyti nuo to, kaip yra vaizduojami, pateikiami sprendiniai, kitaip tariant, nuo sprendinio „kodavimo“ pobudžio. Konkreti GA realizacija pateikiama 4 sk.

4. Tyrimo dalis

4.1. Genetinio algoritmo strukturine schema

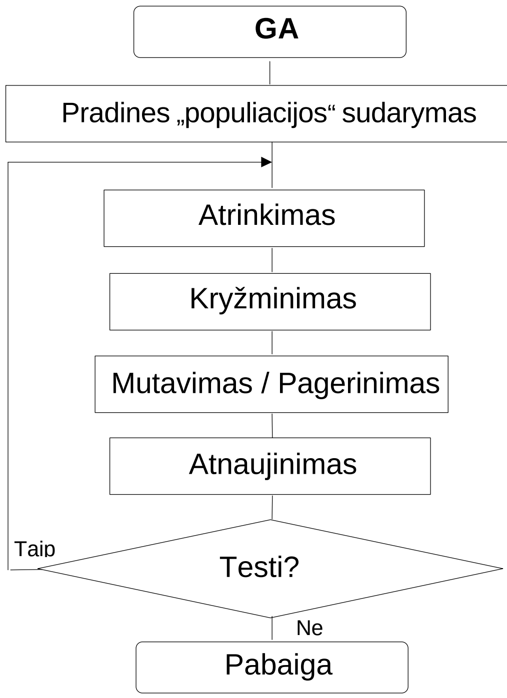
Genetini algoritma galima atvaizduoti strukturine schema (žr. 5 pav.), kuria sudaro tokios dalys:

1. GA algoritmo pradžia.
2. Pradines populiacijos sudarymas.
3. Atskiru „populiacijos“ sprendiniu atrinkimas tolimesniam apdorojimui (pirmenybe teikiama tiems sprendiniams, kurie turi geresne tikslo funkcijos reikšme).
4. Nauju sprendiniu sudarymas, kombinuojant turimus sprendinius. Šis operatorius tradiciškai vadinamas kryžminimu („krosoveriu“).
5. Esant reikalui, panaudojamas papildomas randomizuoto sprendiniu pertvarkymo mechanizmas – mutavimas (Hibridiniuose genetiniuose algoritmuose vietoj mutavimo kartais naudojami lokaliojo sprendinio pagerinimo algoritmai).
6. Atliekamas savotiškas „populiacijos valymas (atnaujinimas)“ (pvz. pašalinant iš jos blogesnius ir paliekant geresnius sprendinius).
7. Tikrinama algoritmo baigimo salyga.
8. Pabaiga.

Algoritmo žingsniai (3)-(7) kartojami tol kol gaunami rezultatai, tenkinantys baigimo salyga.

Nustatyta (eksperimentais), kad „grynasis“ genetinis algoritmas (toks, kaip išdestyta aukšciau) paprastai negali užtikrinti geru, priimtinu rezultatu esant ribotiems kompiuterio centrinio procesoriaus laiko resursams. Kitaip tariant, siekiant gauti aukštos kokybes, artimus optimaliems sprendinius, reikalingi pernelyg ilgi skaiciavimai. Dažnai šie laiko resursai esti žymiai didesni negu panaudojant kitus euristinius algoritmus, pvz. atkaitinimo modeliavima, tabu paieška ir pan. Del šiu priežasciu vietoje standartines GA strukturines schemos taikomos išplestines (hibridizuotos) GA schemos, toliau vadinamos tiesiog hibridiniais genetiniais algoritmais (HGA).

Hibridiniai genetiniai algoritmai labai sekmingai išbandyti sprendžiant daugeli kombinatorinio optimizavimo uždaviniu, tarp ju, kvadratinio paskirstymo uždavini (žr. 2 sk.). HGA principas slypi tame, jog kartu su tradiciniais genetiniais operatoriais panaudojamos papildomos proceduros gaunamu sprendiniu pagerinimui. Tokiu proceduru vaidmenyje galima naudoti ivairius žinomus euristinius algoritmus. Paprastai sprendinio papildomas optimizavimas atliekamas po kryžminimo operacijos, nors galimi ir kiti variantai. Štai dažnai euristines proceduros „tarnauja“ kaip aukštos kokybes pradiniu sprendiniu „populiaciju“ generatoriai. Tokiu budu GA operuoja su optimizuota „populiacija“, o ne su atsitiktiniais pradines „populiacijos“ sprendiniais. Toki genetini procesa galime traktuoti kaip paieška lokaliai optimaliu sprendiniu „erdveje“ – o tai yra daug efektyviau, jeigu lyginti su paieška atsitiktiniu ar tik gana letai gerinamu sprendiniu „erdveje“.



5 pav. Genetinio algoritmo strukturine schema (šablonas)

Yra irodyta, kad „grynasis“ genetinis algoritmas (toks, kaip išdestyta aukšciau) paprastai negali užtikrinti geru, priimtinu rezultatu esant ribotiems kompiuterio centrinio procesoriaus laiko resursams. Kitaip tariant, siekiant gauti aukštos kokybes, artimus optimaliems sprendinius, reikalingi pernelyg ilgi skaiciavimai. Dažnai šie laiko resursai esti žymiai didesni negu panaudojant kitus euristinius algoritmus, pvz. atkaitinimo modeliavima, tabu paieška ir pan. Del šiu priežasciu vietoje standartines GA strukturines schemos taikomos išplestines (hibridizuotos) GA schemos (Moscato, 1999), toliau vadinamos tiesiog hibridiniais genetiniais algoritmais (HGA).

Hibridiniai genetiniai algoritmai labai sekmingai išbandyti sprendžiant daugeli kombinatorinio optimizavimo uždaviniu, tarp ju, kvadratinio paskirstymo uždavini (žr. 2 sk.).

HGA principas slypi tame, jog kartu su tradiciniais genetiniais operatoriais panaudojamos papildomos proceduros gaunamu sprendiniu pagerinimui. Tokiu proceduru vaidmenyje galima naudoti ivairius žinomus euristinius algoritmus. Paprastai sprendinio papildomas optimizavimas atliekamas po kryžminimo operacijos, nors galimi ir kiti variantai. Štai dažnai euristines proceduros „tarnauja“ kaip aukštos kokybes pradiniu sprendiniu „populiaciju“ generatoriai. Tokiu budu GA operuoja su optimizuota „populiacija“, o ne su atsitiktiniais pradines „populiacijos“ sprendiniais. Toki genetini procesa galime traktuoti kaip paieška lokaliai optimaliu sprendiniu „erdveje“ – o tai yra daug efektyviau, jeigu lyginti su paieška atsitiktiniu ar tik gana letai gerinamu sprendiniu „erdveje“.

Efektyvumui padidinti hibridine genetine paieška gali buti kombinuojama su vadinamojo „restarto“ procedura. Šios strategijos esme – rekonstruoti sprendiniu „populiacija“ tais atvejais, kai prarandama „populiacijos“ sprendiniu ivairove. Ivairoves praradimas yra negatyvus faktorius, salygojantis per daug ankstu genetinio proceso konvergavima i visai nebutinai aukštos kokybes lokaluji optimuma; tuo paciu pasireiškia vadinamoji paieškos „stagnacija“, t.y. toks fenomenas, kai po tam tikro laiko – kad ir kiek paieška besitestu – neimanoma (arba labai sunku tiketis) surasti nauju geresniu. Realizuojant „restarta“ paprasciausiu atveju pakanka tiesiog iš naujo sugeneruoti sprendiniu „populiacija“ (gal but, tik paliekant iš buvusios „populiacijos“ viena geriausia sprendini), nors galimi ir labiau rafinuoti budai. Perspektyvi yra „populiacijos“ rekonstravimo strategija, pagrista specialios mutavimo proceduros taikymu „populiacijos“ sprendiniams. Mutavimas neturetu buti nei per „stiprus“, nei per „silpnas“: pirmuoju atveju genetine paieška taptu panaši i daugkartine atsitiktinio starto paieška; antruoju – nebutu garantuojamas pakankamai „nutolusiu“ (nuo esamu) sprendiniu generavimas. Detalizuotas HGA KP uždaviniams šablonas pateiktas 6 pav.

Magistro darbo tikslas – išnagrineti ivairias HGA modifikacijas, kurios skiriasi sprendiniu kryžminimo operatoriais („krosoveriais“). Toliau ir nagrinejamos kryžminimo operatoriu modifikacijos.

```
procedure hibridinis_genetinis_algoritmas;
//pradiniai duomenys: PD – sprendiniu „populiacijos“ dydis //
//rezultatai: s* – geriausias rastas sprendinys //
begin
  sukurti pradine optimizuota „populiacija“ P⊂S (|P|=PD) panaudojant
  viena ar daugiau konstrukciniu/euristiniu algoritmu;
  s* ← argmin_{s∈P} f(s) ;  // isimenamas geriausias „populiacijos“ sprendinys //
  repeat // vykdomas pagrindinis (generaciju) ciklas //
    repeat // vykdomas „sprendiniu-palikuoniu“ formavimo ciklas //
      parinkti "sprendinius-tevus" s′,s″∈ P ;
      s‴ := krosoveris(s′,s″); // „tevams“ atliekamas kryžminimas, gaunant „sprendini-palikuoni“ //
      pagerinti sprendini s‴ panaudojant viena iš euristiniu algoritmu;
      P := P ∪ {s‴}; // pagerintas „palikuonis“ itraukiamas i „populiacija“ //
      if f(s‴) < f(s*) then s* := s‴  // isimenamas geriausias rastas sprendinys //
    until turi buti pereinama i sekancia generacija;
    atnaujinti „populiacija“ P, pvz. atmetant blogesnius ir paliekant
    geresnius sprendinius;
    esant reikalui, vykdyti "restarta" („populiacijos“ P sprendiniu
    rekonstravima)
  until patenkinta baigimo salyga
end.
```

6 pav. Hibridinio genetinio algoritmo šablonas

4.2. Kryžminimo operatoriu modifikacijos

Kryžminimo procedura - viena iš pagrindiniu genetines paieškos operaciju. Ji geba sugeneruoti galima sprendini iš tevus apibudinancios informacijos. Krosoveris yra griežtai strukturiškai apibrežtas ir tuo paciu metu atsitiktinis procesas, kuris užtikrina abieju tevu charakteristiku paveldejima ir nauju savybiu sukurima. Matematiškai kryžminimo operacija gali buti aprašoma kaip binarine funkcija y: $\Pi \times \Pi \rightarrow \Pi$ tokia, jog $y(p', p'') \neq p' \vee y(p', p'') \neq p''$ jeigu $p' \neq p''$; cia mes laikome, jog sprendinys yra perstatymas. Kaip taisykle, visos kryžminimo proceduru modifikacijos turi užtikrinti, jog palikuoniai neabejotinai paveldetu abiems tevams bendras charakteristikas. Formaliai galima butu tai užrašyti taip: $p'(i) = p''(i) \Rightarrow p^{\circ}(i) = p'(i) = p''(i)$, $i = 1, 2, ..., n$. Likusiu charakteristiku (genu) paveldimumas gali buti realizuojamas ivairiais budais.

Kryžminimo operatoriu nagrinejima pradedame nuo kryžminimo operacijos, kuria 1995 metais pasiule Tate ir Smith [31]. Šis krosoveris gali buti laikomas tolygiojo pasiskirstymo krosoveriu rušies, be to yra adaptuotas perstatymams. Toliau ji vadinsime tolygiojo pasiskirstymo krosoveriu (ULX – *angl. uniform like crossover*). Šios kryžminimo operacijos veikimo principas aprašomas taip:

1. Visos vienodos tevu charakteristikos tose paciose pozicijose nukpijuojamos i ta pacia pozicija palikuonyje.

2. Pozicijos, kuriose nepriskirtos charakteristikos, analiuzuojamos iš kaires i dešine: tušciose pozicijose reikšme parenkama atsitiktinai iš kurio nors tevo, jeigu tik tokia reikšme nebuvo priskirta palikuoniui anksciau.

3. Likusios pozicijos užpildomos atsitiktinai.

3	6	7	4	1	5	2	9	8	tevas 1
7	3	6	4	2	9	5	1	8	tevas 2
			4					8	
7	6		4	2	5		1	8	
3	9								
7	6	9	4	2	5	3	1	8	palikuonis

7 pav. Tolygiojo pasiskirstymo krosoverio pavyzdys

Tolygiojo pasiskirstymo kryžminimo operacija (ULX) yra labia lanksti ir galimos ivairios algoritmo variacijos. Toliau pateikiamos trys klasikines ULX proceduros modifikacijos. Kai kurias atvejais modifikacijos gali buti laikomos naujomis krosoverio rušimis. Jas pavadinsime taip: Atsitiktinis ULX (RULX – *angl. randomized ULX*), modifikuotas ULX arba blokinis krosoveris (BX – *angl. block crossover*) ir išplestinis ULX arba koreguojantis krosoveris (RX – *angl. repair crossover*).

Esminis skirtumas tarp standartinio tolygiojo pasiskirstymo krosoverio (ULX) ir atsitiktinio (RULX) yra metode, kuris analizuoja charakteristiku pozicijas sprendinyje. Standartiniame ULX algoritme poziciju analizavimo tvarka yra fiksuota – iš kaires i dešine. Atsitiktiniame ULX algoritme pozicijos parenkamos atsitiktinai. Taip palikuonims suteikiama daugiau atsitiktinumo ir ivairumo, tuo užtikrinant mažesne algoritmo priešlaikinio konvergavimo tikimybe.

Kita standartinio ULX algoritmo modifikacija – vadinama blokinis krosoveris (BX) – išsiskiria tuo, jog vietoj pavieniu elementu naudojami keliu elementu blokai (segmentai). Vieno bloko dydis (*BD*) yra $[1, \lfloor n/2 \rfloor]$ intervalo ribose. Reiktu pastebeti, jog bloku dydis gali buti skirtingas; pavyzdžiui uždavinio apimtis n=9 ir naudosime 4 blokus. Tuomet triju bloku dydis bus 2 elementai, ir vienas blokas bus iš 3 elementu. Kopijuojant blokus turi buti „išlaikytas“ perstatymas (žiureti 8 pav.). $BD = \lfloor n/2 \rfloor$ reiškia, jog pirmas segmentas, kurio dydis $\lfloor n/2 \rfloor$, kopijuojamas palikuoniui iš vieno tevo (tarkime, p'). Like blokai kopijuojami iš „šalia“ esancio tevo (p'') tokiu budu, jog butu išlaikytas galutinio sprendinio taisyklingumas. Jeigu lieka nepriskirtu poziciju palikuonyje, tuomet trukstamos reikšmes gali buti parinktos iš bet kurio tevo.

3	6	7	4	1	5	2	9	8	Tevas 1
7	3	6	4	2	9	5	1	8	Tevas 2
7	3		4	1	9	5		8	
6	2								Trukstamos reikšmes
7	3	6	4	1	9	5	2	8	Palikuoni

8 pav. Blokinio krosoverio pavyzdys

Esmine išplestinio ULX krosoverio ideja yra klasikinio ULX ir pataisymo proceduros, pagristos lokaliu pagerinimu, kombinacija. Labiau sukonkretinat, bandoma pagerinti palikuoni atliekant elementu porini sukeitima, bet tik tu elementu, kurie nera paveldeti iš tevu. Sukuriamas kandidatu sarašas (KS), kur $KS(i) = p^{\circ}(i)$, $KS(i) \neq p'(i)$, $KS(i) \neq p''(i)$ (p', p'', p° - atitinkamai tevai ir palikuonis). Kandidatu sarašo nariai užima tam tikra dali gerinimo proceso eigoje (žinoma, jeigu kandidatu sarašas yra tušcias ($|KS| = 0$), gerinimo procesas neatliekamas). Aprašytos proceduros šablonas pateiktas 9 pav.

procedure *dalinis_staigus_nusileidimas*

// Tarkime p - palkuonis. Kiekvieno proceduros žingsnio metu,

// sprendinys pakeiciamas nauju sprendiniu kuris pagerina tikslo

// funkcijos reikšme.

// Procesas kartojamas kol gerinamas sprendinys egzistuoja

repeat

$$p^{\bullet} := \underset{\substack{i=CL(1),\ldots,CL(w-1)\\ j=i+1,\ldots,CL(w)}}{\arg\min} z(p \oplus p_{ij})\ ;$$

if $z(p^{\bullet}) < z(p)$ **then** $p := p^{\bullet}$ // Pakeiciamas senas sprendinys nauju

until $p^{\bullet} \neq p$

end // *dalinis_staigus_nusileidimas*

9 pav. Dalinio lokalaus pagerino algoritmo, kuris naudojamas koreguojanciame krosoveryje, šablonas.

Pastaba nr. 1. p_{ij} – elementari perstatymo operacija, kuri paprašciausiai sukeicia i-taji ir j-taji elementus sprendinyje; šiuo atveju, išraiška $p \oplus p_{ij}$ reiškia perstatyma, kuris atliekamas sprendiniui p pritaikant sukeitimo operacija p_{ij}.

Pastaba nr. 2. $w = |KS|$

Atlikus eksperimentus, paaiškejo, jog daugeliu atveju kandidatu sarašo (KS) dydis žymiai mažesnis nei n (uždavinio apimtis). Todel dalinio lokalaus pagerinimo procedura nereikalauja labai dideliu laiko sanaudu.

Toliau aprašysime gerai žinomos kombinavimo (*angl. recombination operator*) proceduros variacija – dalinio atvaizdavimo krosoveri (PMX - *angl. partially-mapped crossover*). Pagrindine ideja šios proceduros yra ta, jog ji dirba tik su chromosomu dalimi – atvaizdavimo sritimis – pasiskirsciusiose tarp dvieju krosoverio tašku [18]. Buvo irodyta, jog PMX labai efektyvus sprendžiant komivojažieriaus uždavini [18], taciau standartine PMX procedura neefektyvi sprendžiant kvadratinio pasiskirstymo uždavini. Del šios priežasties, Migkikh ir kt. [24] pasiule PMX algoritmo modifikacija, kuri vietoj vieno atvaizdavimo segmento naudoja keleta atsitiktinio atvaizdavimo tašku. Ši kryžminimo procedura vadinama tolygiuoju PMX (UPMX - *angl. uniform PMX*). Pagrindiniai UPMX etapai yra šie:

1) Klonuojamas palikuonis $p°$ iš pirmo tevo p'.

2) Atsitiktinai palikuonyje parenkama atsitiktine pozicija pos_1.

3) Randama palikuonyje pozicija pos_2, kurios turinys (reikšme) yra lygus antro tevo p'' turiniui (reikšmei), esanciam pozicijoje pos_1. t. y. $p°(pos_2) = p''(pos_1)$.

4) Sukeiciamos tarpusavyje $p°(pos_1)$ ir $p°(pos_2)$ reikšmes.

5) Kartojame nuo pradžiu k kartu, kur $k = \lfloor n/3 \rfloor$ (pagal [24]).

Algoritmo principas pateikiamas 10 pav.

1	**8**	**6**	**2**	**4**	**5**	**3**	**7**	**9**	Tevas 1
6	**7**	**1**	**4**	**2**	**9**	**8**	**5**	**3**	Tevas 2
1	8	6	2	4	5	3	7	9	Pirmo tevo klonas
1	8	6	2	4	5	3	7	9	Žingsnis nr. 1
1	3	6	2	4	5	8	7	9	Žingsnis nr. 2
6	3	1	2	4	5	8	7	9	Žingsnis nr. 3
6	**3**	**1**	**2**	**4**	**7**	**8**	**5**	**9**	Palikuonis

10 pav. Tolygiojo dalinio planavimo krosoverio pavyzdys

Merz ir Freisleben [23] pasiule „išlaikancia atstuma“ kryžminimo operacija (DPX – *angl. distance preserving crossover*) pritaikyta specialiai kvadratinio pasiskirstymo uždaviniams spresti. (Ankstesnes šio algoritmo versijos buvo bandomos sprendžiant komivojažieriaus uždavini). Pagrindine šio algoritmo ideja yra ta, jog kombinavimo procedura stengsis, „kuriant“ sprendini, išsaugoti vienoda „atstuma“ iki tevu. Šis „atstumas“ yra lygus „atstumui“ tarp tevu.

„Atstumas“ r tarp dvieju perstatymu p_1 ir p_2 nusakomas elementu, kurie yra paskirti i skirtingas pozicijas, skaiciumi. T. y. $r(p_1, p_2) = |\{i \mid p_1(i) \neq p_2(i)\}|$. Taigi, elementai, kurie yra vienodi tose paciose tevu (p' ir p'') pozicijose, kopijuojami i palikuoni ($p°$). Visi kiti genai yra sukeiciami; t. y. likusios pozicijos atsitiktinai užpildomos dar nepriskirtomis reikšmemis, užtikrinant, jog priskirta reikšme konkrecioje pozicijoje nesutaptu su tevu reikšmemis toje pozicijoje. DPX krosoverio pavyzdys pateikiamas 11 pav.

3	**5**	**8**	**2**	**9**	**1**	**4**	**6**	**7**	Tevas 1
8	**5**	**6**	**4**	**9**	**1**	**3**	**2**	**7**	Tevas 2
	5			9	1			7	
6		2	3			8	4		
6	**5**	**2**	**3**	**9**	**1**	**8**	**4**	**7**	Palikuonis

$$r(p',p^{\circ}) = r(p'',p^{\circ}) = r(p',p'') = 5$$

11 pav. „Išlaikancio atstuma“ krosoverio pavyzdys

Toliau apžvelgsime ciklinio kryžminimo (CX – *angl. cycle crossover*) operacijos principus. Esminiai šio algoritmo esminis principas yra tas, jog stengiamasi išsaugoti abiejuose tevuose esama informacija. T. y. visos palikuonio reikšmes yra gautos iš kurio nors tevo. Pagrindiniai ciklinio krosoverio žingsniai butu šie:

1) Visos reikšmes, rastos tose paciose pozicijose abiejuose tevuose, priskiriamos atitinkamoje pozicijoje palikuoniui.

2) Pradedant nuo pirmos pozicijos (arba atsitiktinai prinktos) (atsižvelgiant i tai, jog analizuojamas elementas dar nera priskirtas palikuoniui), elementas atsitiktinai parenkamas iš vieno iš tevu. Atliekamas papildomas veiksmas, užtikrinantis, jog neivyktu atsitiktiniu priskyrimu (mutaciju). Tas pats kartojama su sekancia tušcia pozicija ir kartojama tol kol bus užpildytos visos pozicijos palikuonyje. Pavyzdys pateikiamas 12 pav.

3	**5**	**8**	**2**	**9**	**1**	**4**	**6**	**7**	Tevas 1
8	**5**	**6**	**9**	**4**	**1**	**2**	**3**	**7**	Tevas 2
	5				1			7	Žingsnis nr. 1: elementai 5,1,7 paveleti iš tevu
3		8					6		Žingsnis nr. 2: pradine pozicija 3, elementas 8
			9	4		2			Žingsnis nr. 3: pradine pozicija 4, elementas 9
3	**5**	**8**	**9**	**4**	**1**	**2**	**6**	**7**	Palikuonis

12 pav. Ciklinio krosoverio pavyzdys

Ahuja ir kt. “godžiuose genetiniuose algoritmuose“ [1] naudoja poriniu sukeitimu krosoveri (SPX – *angl. swap path crossover*). (Reiketu pastebeti, jog pagrindine šio algoritmo ideja pirmasis pasiule Glover [14], kuris ivardino tai kaip „trajektoriju peradresavimas“). Tarkime p', p'' yra tevu pora. Pradedama nuo pirmo (arba atsitiktinai parinkto) geno ir tevai tikrinami iš kaires i dešine tol kol visi genai bus ivertinti. Jeigu reikšme tose paciose pozicijose yra vienoda, tuomet pereiname i sekancia pozicija; priešingu atveju atliekamas elementu sukeitimas abiejuose tevuose (p' ir p''). Pavyzdžiui analizuojamas genas yra i ir $a = p'(i)$, $b = p''(i)$, tuomet po sukeitimo arba $p'(i)$ igauna b, arba $p''(i)$ igauna a reikšme. Ahuja et al. pasiule atlikti sukeitima tam sprendiniui, kurio verte mažesne (tikslo funkcijos reikšme geresne). Elementai dviejuose gautuose sprendiniuose toliau nagrinejami pradedant nuo sekancios pozicijos ir taip toliau. Geriausias sprendinys šio proceso eigoje (tinkamiausias vaikas) tampa palikuoniu. Poriniu sukeitimu krosoverio algoritmo fragmentas iliustruojamas 13 pav.

5	**2**	**3**	**4**	**1**	**7**	**6**	**8**	**9**	Tevas 1
2	**1**	**3**	**4**	**6**	**5**	**7**	**8**	**9**	Tevas 2
2	5	3	4	1	7	6	8	9	vaikas 1
5	1	3	4	6	2	7	8	9	vaikas 2
2	5	3	4	1	7	6	8	9	Geriausias vaikas
2	**5**	**3**	**4**	**1**	**7**	**6**	**8**	**9**	Nauja tevu pora
2	**1**	**3**	**4**	**6**	**5**	**7**	**8**	**9**	
2	1	3	4	5	7	6	8	9	vaikas 3
2	5	3	4	6	1	7	8	9	vaikas 4
...	...	...	...	...	...	...	...	...	

palikuonis = geriausias

13 pav. Poriniu sukeitimu krosoverio pavyzdys

Vieno taško krosoveris (OPX – *angl. one point crossover*) [17] yra klasikines kombinavimo proceduros variantas, kuris buvo placiai taikomas ankstyvuosiuose genetiniuose algoritmuose. Viena OPX algoritmo modifikacija buvo pasiulyta Lin et. al. [21]. Šios proceduros pagrindine ideja pakankamai paprasta. Dalijimo taškas viename iš tevu pasirenkamas atsitiktinai tarp 1 ir n – 1 poziciju. Galutinis sprendinys (palikuonis) gaunamas sukeitus tevu chromosomas tarpusavyje, išlaikant galutinio sprendinio taisyklinguma. Algoritmo principas pavaizduotas 14 pav.

Dalijimo taškas

2	**8**	**7**	**1**	**4**	**5**	**9**	**3**	**6**	Tevas 1
7	**6**	**4**	**1**	**8**	**9**	**2**	**3**	**5**	Tevas 2
2	8	7	1	4	5				Elementai tevo 1
						6	3	9	Elementai tevo 2
2	**8**	**7**	**1**	**4**	**5**	**6**	**3**	**9**	Palikuonis

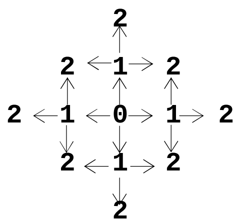
14 pav. Vieno taško krosoverio pavyzdys

Taip vadinamas „paremtas tvarka“ operatorius [8] buvo sekmingai pritaikomas sprendžiant tvarkarašciu sudarymo uždavinius, kur tikslas yra nustatyti tvarka (seka) kuria darbai priskiriami pagal grafika. Pagrindine „tvarka paremto“ krosoverio (OBX – *angl. order-based crossover*) savybe yra ta, jog išsaugoma genu tvarka chromosomoje. Taigi, keletas elementu parenkami iš vieno iš tevu ir nukopijuojami i palikuoni. Trukstami elementai, laikantis tvarkos, nukopijuojant iš kito tevo. Šio krosoverio pavyzdys pateikiamas 15 pav.

8	6	4	2	1	5	9	3	7	Tevas 1
2	3	4	6	7	1	5	9	8	Tevas 2
0	1	0	1	1	0	0	0	1	Atsitiktine maske
	6		2	1				7	Elementai tevo 1
3		4			5	9	8		Trukstami elementai
3	6	4	2	1	5	9	8	7	Palikuonis

15 pav. Tvarka paremto krosoverio pavyzdys

Neseniai, Drezner‘is pasiule ganetinai originalu kombinavimo operatoriaus sprendima – surišanti kryžminimo operatoriu [10] (COHX – *angl. cohesive crossover*). Šiame algoritme palikuonis gaunamas, atlikus kelis žingsnius. Pradžioje sukuriama atstumu matrica $\boldsymbol{M}$, kurios dydis $n_1 \times n_2$. n_1 ir n_2 parenkami tokie, jog tenkintu salyga: $n_1 \cdot n_2 = n \wedge n_1 + n_2 \rightarrow \min$. Fiksuojama pradine padetis atstumu matricoje (i_0, j_0), kur $i_0 \in \{1, 2, ..., n_1\}$, $j_0 \in \{1, 2, ..., n_2\}$. Atstumu matrica užpildoma pagal bangos sklidimo principa (žiur. 16 pav.).



16 pav. Atstumu matricos pildymas

Egzistuoja n skirtingu atstumu matricos variantu: $\boldsymbol{M}^{(1)}, \boldsymbol{M}^{(2)}, ..., \boldsymbol{M}^{(k)}, ..., \boldsymbol{M}^{(n)}$, kur k, i_0, ir j_0 yra ryšyje: $k = n_2 \cdot (i_0 - 1) + j_0$, $i_0 = 1, 2, ..., n_1$, $j_0 = 1, 2, ..., n_2$. Tuomet k-tasis sprendinys $p^{(k)}$ $(k \in \{1, 2, ..., n\})$ gaunamas atliekant šiuos žingsnius:

1) $p^{(k)}(i) = \begin{cases} p_{geresnis}(i), \boldsymbol{M}^{(k)}(((k-1) \text{ div } n_2) + 1, ((k-1) \text{ mod } n_2) + 1)) \le h \\ 0, \text{priešingu atveju} \end{cases}$;

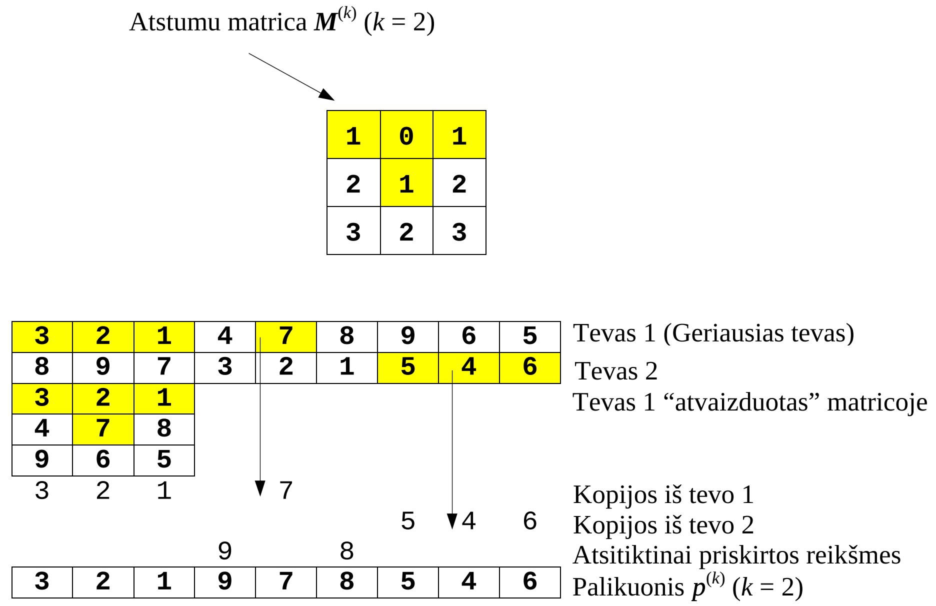
kur $i = 1, 2, ..., n$, $p_{geresnis} = \arg\min \{z(p'), z(p'')\}$, p', p'' yra sprendiniai - tevai, ir h yra matricos $\boldsymbol{M}^{(k)}$ mediana, t. y. $h = \dfrac{\sum_{i}^{n_1}\sum_{j}^{n_2} \boldsymbol{M}^{(k)}(i,j)}{n_1 \cdot n_2}$;

$$2)\ p^{(k)}(i) = \begin{cases} p^{(k)}(i), p^{(k)}(i) > 0 \\ p_{b\log esnis}(i), p^{k}(i) = 0 \wedge p_{b\log esnis}(i)\ nepriklauso\ p^{(k)} \\ 0, \text{priešingu atveju} \end{cases};$$

kur $i = 1, 2, ..., n$, $p_{blogesnis} = \arg\max \{z(p'), z(p'')\}$;

3) kiekvienai nepriskirtai pozicijai i ($p^{(k)}(i) = 0$), reikšme parenkama atsitiktinai iš reikšmiu, kurios dar nera priskirtos palikuoniui.

Sprendinio generavimo pavyzdys pateikiamas 17 pav.



17 pav. Surišancio krosoverio pavyzdys

Drezner pasiulytas algoritmas yra pakankamai lankstus ir galimos ivairios jo modifikacijos. Vienoje iš modifikaciju siuloma generuoti kvadratine atstumu matrica. Atsiranda vienintele problema, jog uždavinio apimtis ne visuomet sekmingai „atvaizduojama“ i kvadratine matrica. Todel siuloma parinkti toki matricos kraštines dydi n_d, jog $\mathrm{abs}(n_d^2 - n) \rightarrow \min$. Savaime suprantama, jog tokiu atveju liks matricoje nepanaudojamu elementu. Tuo tikslu, sukuriama bitine kaukes matrica $\boldsymbol{K}$ $n_d \times n_d$, kurioje pažymimi naudojami ir nenaudojami matricos elementai. Pastarieji išdestomi paskutiniame matricos stulpelyje ir paskutineje eiluteje pradedant nuo (n_d, n_d) elemento. Tarkime turime uždavini, kurio apimtis n=11. Tada $n_d = 4$. Kaip atrodys suformuota bitine kaukes matrica, pavaizduota 18 pav.

1	1	1	1
1	1	1	0
1	1	1	0
1	0	0	0

18 pav. Bitine kaukes matrica ***K***

Tolimesni algoritmo veiksmai atliekami atsižvelgiant i bitines kaukes matricos, konkrecios pozicijos reikšme. T.y. esant reikšmei 1 – skaiciavimai atliekami, esant 0 – praleidžiami. Šio algoritmo pavyzdys, kai uždavinio apimtis n = 11 pateikiamas žemiau:

Bitine kaukes matrica ***K***

1	1	1	1
1	1	1	0
1	1	1	0
1	0	0	0

Atstumu matrica $\boldsymbol{M}^{(k)}$ (k = 10)

4	3	2	3
3	2	1	
2	1	0	
3			

3	2	1	4	7	8	11	6	10	9	5	Tevas 1 (Geriausias tevas)
2	9	7	3	5	11	10	4	8	1	6	Tevas 2

3	2	1	4	Tevas 1 “atvaizduotas” kaukes matricoje
7	8	11		
6	10	9		
5				

		1			8	11	6	10	9		Kopijos iš tevo 1
2			3	5							Kopijos iš tevo 2
	4									7	Atsitiktinai priskirtos reikšmes
2	4	1	3	5	8	11	6	10	9	7	Palikuonis $p^{(k)}$ (k = 10)

19 pav. Modifikuoto surišancio krosoverio pavyzdys

Keleto tevu kryžminimo procedura (MPX – *angl. multiple parent crossover*) apraše Misevicius [25], nors ideja naudoti keleta sprendiniu pasiule Boese ir kt. [4] ir Fleurent ir Glover [13]. Algoritmas MPX pasižymi tuo, jog palikuonis paveldi informacija iš keletos tevu. Tai priešingybe ir šiuo metu pranašumas prieš tradicini kryžminima, kuri atliekant, naudinga informacija gali buti praleista, nes naudojami tik du tevai. MPX algoritme *i*-tasis elementas, pvz. chromosoma iš palikuonio p° sukuriamas pasirinkus toki skaiciu j (iš nepanaudotu reikšmiu), kurio tikimybe, kad $p°(i) = j$ $Tk(p°(i) = j)$ yra maksimizuota. Cia tikimybe $Tk(p°(i) = j)$ yra lygi $\dfrac{d_{ij}}{\sum_{j}^{n} d_{ij}}$, kur d_{ij} – elementas iš „siekio“ matricos $\boldsymbol{D} = (d_{ij})_{n\times n}$.

Procesas kartojamas tol, kol visi palikuonio genai igaus reikšmes. Šio algoritmo pavyzdys pateikiamas 20 pav.

4	**3**	**6**	**7**	**1**	**2**	**9**	**8**	**5**
4	**3**	**6**	**7**	**1**	**9**	**5**	**8**	**2**
4	**6**	**3**	**1**	**7**	**5**	**9**	**2**	**8**
4	**7**	**3**	**1**	**8**	**5**	**9**	**6**	**2**
5	**6**	**3**	**1**	**2**	**4**	**9**	**7**	**8**

Penki tevai

0	0	0	4	1	0	0	0	0
0	0	2	0	0	2	1	0	0
0	0	3	0	0	2	0	0	0
3	0	0	0	0	0	2	0	0
2	1	0	0	0	0	1	1	0
0	1	0	1	2	0	0	0	1
0	0	0	0	1	0	0	0	4
0	1	0	0	0	1	1	2	0
0	2	0	0	1	0	0	2	0

„Siekio“ matricos elementai

Tarkime, jog indeksu seka yra tokia:

7, 3, 1, 8, 2, 6, 5, 4, 9;

Tuomet palikuonis yra kuriamas taip:

$p(7) = \arg\max_{j}\{\Pr(p(7) = j)\} = \arg\max_{j}\{d_{7j}\} = 9$;

$p(3) = \arg\max_{j\neq 9}\{\Pr(p(3) = j)\} = 3$; $p(1) = \arg\max_{j\neq 3,9}\{\Pr(p(1) = j)\} = 4$;

$p(8) = 8$; ir t.t.

4	**6**	**3**	**7**	**1**	**5**	**9**	**8**	**2**

Palikuonis

20 pav. Keleto tevu krosoverio pavyzdys

Reiketu pamineti, jog šiame darbe „siekio“ matrica yra šiek tiek modifikuota. Vietoj $\boldsymbol{D} = (d_{ij})_{n\times n}$, naudojame $\boldsymbol{D}$¢ $= (d'_{ij})_{n\times n}$, kur $d'_{ij} = d_{ij} + e$, cia e pataisos (triukšmo) koeficientas.

Visi šia aptarti algoritmai, gali buti skirstomi ivairiomis kategorijomis atsižvelgiant i tam tikrus kriterijus: tevu skaiciu, atsitiktinumo lygi (iškraipymus), pritaikymas konkrecios problemos sprendimui, laiko ir atminties apribojimai, programavimo aspektai. Pabandysiu detaliau aptarti atsitiktinumo faktoriu. Šiurkšciai tariant, šis faktorius gali buti vertinamas kaip matas, kuris nurodo kiek palikuonis „atitoles“ nuo tevu. Pasinaudojant šiuo faktorium, krosoverio operacijos gali buti klasifikuojamos kaip nedaug griaunancios ir daug griaunancios. Radciffe ir Surry [27] atitinkamai naudoja terminus „pilnai išlaikantis tvarka“ (*angl. assorting*) ir „nepilnai išlaikantis tvarka“ (*angl. respectful*). (Reiktu pastebeti, jog daugelis analizuotu krosoveriu gali buti laikomi „nepilnai išlaikanciais tvarka“, tik ciklinis krosoveris papuola i „pilnai išlaikanciu tvarka“ algoritmu kategorija. Pagrinde yra dvi situacijos suliejimo procese: naturali mutacija ir iššaukiama mutacija. „Pilnai išlaikantys tvarka“ kryžminimo algoritmai charakterizuojami naturalia mutacija, o „nepilnai išlaikantys tvarka“ algoritmai charakterizuojami iššaukiama mutacija. Naturalios mutacijos situacija yra tada, kai palikuonis skiriasi nuo abieju tevu, taciau kiekvienas vaiko elementas yra iš atitinkamo geno iš kurio nors tevo. T.y. $p° \neq p' \wedge p° \neq p'' \wedge (p°(i) = p'(i) \vee p°(i) = p''(i)$, $i = 1, 2, ..., n)$. Tai labai griežta salyga. Del šios priežasties „grupuojantys“ algoritmai sunkiai realizuojami nekuriems uždaviniams. Tuo tarpu iššaukiama mutacija suteikia daugiau laisves. Vienintelis reikalavimas - patenkinti salyga, jog palikuonis skirtusi nuo tevu (numatant, jog palikuonis neišvengiamai paveldes bendras tevu chromosomas). Labiau konkretinant, numatoma mutacija laikoma tuomet, kai egzistuoja (bent vienas) toks i, kuris $p°(i) \neq p'(i) \wedge p°(i) \neq p''(i)$. Elementas gali buti laikomas „svetimu“, jeigu jo nera bent viename iš tevu. Svetimu elementu kiekis gali buti laikomas kryžminimo algoritmo „griovimo“ rodikliu.

5. Eksperimentu rezultatai

Algoritmu efektyvumo tyrimams pasinaudota kvadratinio pasiskirstymo uždavinio testiniais pavyzdžiais (duomenimis) iš pavyzdžiu bibliotekos QAPLIB [6]. Tarp daugelio pavyzdžiu išskirti atsitiktiniu budu sugeneruoti pavyzdžiai ir „realaus pasaulio“ duomenis imituojantys pavyzdžiai. Pirmajai grupei priklauso šie konkretus pavyzdžiai: tai25a, tai30a, tai35a, tai40a, tai50a, tai60a, tai80a ir tai100a (tai∗a), o antrajai – pavyzdžiai: tai25b, tai30b, tai35b, tai40b, tai50b, tai60b, tai80b, tai100b ir tai150b (tai∗b). Eksperimentai buvo vykdyti su programa OPTIQAP (OPTImizer for the QAP) ir 2,6 GHz taktinio dažnio PENTIUM kompiuteriu.

Buvo pasirinkti šie algoritmu ivertinimo kriterijai:

a) vidutinis nuokrypis nuo geriausio žinomo sprendinio – $\bar{d}$ ($\bar{d} = 100(\bar{z} - z_{ger})/z_{ger}$ [%], cia $\bar{z}$ yra tikslo funkcijos reikšmiu vidurkis, apskaiciuotas, atlikus W (W = 10) algoritmo pakartotiniu vykdymu, o z_{ger} yra geriausia žinoma (tikslo funkcijos) reikšme (GŽR));

b) sprendiniu, esanciu „1% optimalumo intervale“, skaicius;

c) surastu geriausiu žinomu (pseudo-optimaliu) sprendiniu skaicius.

Atlikta 4 eksperimantai su skirtingais testiniais pavyzdžiais ir generaciju skaiciumi:

Lentele nr. 1 Eksperimentu aprašymas

Pavyzdžiai / Genera-ciju skaicius	Atsitiktinai sugeneruoti	Realaus pasaulio
7	I	II
20	III	IV

Eksperimentu rezultatai pateikiami sekanciose lentelese.

Lentele nr. 2 Eksperimento I rezultatai (1 dalis)

		ULX			RULX			RX			BX			UPMX		
		Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis
	TAI020A	0.713	10	1	0.615	10	1	0.666	10	1	0.73	10	1	0.68	10	1
	TAI025A	0.913	7	0	0.884	7	0	0.71	8	0	1.005	5	0	0.754	7	1
	TAI030A	0.682	10	0	0.652	10	0	0.648	9	0	0.685	8	0	0.525	9	0
	TAI035A	0.778	8	1	0.671	9	1	0.734	8	1	0.754	8	1	0.659	9	1
I	TAI040A	0.775	10	0	0.802	8	0	0.81	8	0	0.727	9	0	0.838	7	0
	TAI050A	0.913	7	0	0.952	6	0	0.918	8	0	0.964	6	0	0.961	6	0
	TAI060A	0.873	9	0	0.845	9	0	0.866	9	0	0.833	10	0	0.876	9	0
	TAI080A	0.519	10	0	0.492	10	0	0.498	10	0	0.499	10	0	0.506	10	0
	TAI100A	0.411	10	0	0.411	10	0	0.411	10	0	0.377	10	0	0.402	10	0

Lentele nr. 3 Eksperimento I rezultatai (2 dalis)

		SPX			CX			DPX			OPX			OBX		
		Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis
	TAI020A	0.622	10	1	0.652	10	1	0.718	10	1	0.622	10	1	0.681	10	1
	TAI025A	0.899	7	0	0.923	6	0	1.09	4	0	0.717	8	2	0.882	8	0
	TAI030A	0.668	9	0	0.603	10	0	0.827	7	0	0.531	10	0	0.693	9	1
	TAI035A	0.814	6	1	0.695	8	1	0.779	7	1	0.763	8	1	0.711	8	1
I	TAI040A	0.765	10	0	0.786	8	0	0.863	7	0	0.752	10	0	0.77	8	0
	TAI050A	0.865	8	0	0.901	8	0	0.966	7	0	0.945	7	0	0.919	7	0
	TAI060A	0.828	10	0	0.833	9	0	0.878	9	0	0.85	10	0	0.841	10	0
	TAI080A	0.435	10	0	0.458	10	0	0.522	10	0	0.458	10	0	0.518	10	0
	TAI100A	0.314	10	0	0.37	10	0	0.406	10	0	0.377	10	0	0.403	10	0

Lentele nr. 4 Eksperimento I rezultatai (3 dalis)

		MPX			COHX1			COHX 2			COHX 3			COHX 4		
		Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis
I	TAI020A	0.615	10	1	0.596	10	2	0.708	10	1	0.409	10	2	0.35	10	2
	TAI025A	0.681	9	1	0.966	6	0	0.871	7	0	0.478	9	2	0.4	10	2
	TAI030A	0.634	9	0	0.717	8	0	0.657	10	0	0.446	10	1	0.454	10	1
	TAI035A	0.681	9	1	0.783	9	1	0.767	7	1	0.553	9	0	0.528	10	0
	TAI040A	0.72	10	0	0.797	8	0	0.744	9	0	0.609	10	0	0.602	10	0
	TAI050A	0.881	8	0	0.887	9	0	0.937	6	0	0.835	8	0	0.865	9	0
	TAI060A	0.814	10	0	0.834	10	0	0.811	10	0	0.765	10	0	0.775	9	0
	TAI080A	0.45	10	0	0.448	10	0	0.438	10	0	0.432	10	0	0.432	10	0
	TAI100A	0.326	10	0	0.389	10	0	0.368	10	0	0.288	10	0	0.272	10	0

Lentele nr. 5 Eksperimento II rezultatai (1 dalis)

		ULX			RULX			RX			BX			UPMX		
		Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis
II	TAI020B	0.045	10	9	0.09	10	8	0.045	10	9	0.091	10	8	0.09	10	8
	TAI025B	0.114	10	3	0.15	10	2	0.201	10	2	0.243	9	3	0.194	10	2
	TAI030B	0.466	8	1	0.315	9	0	0.41	9	2	0.596	8	0	0.549	8	0
	TAI035B	0.237	10	3	0.289	10	0	0.287	10	1	0.277	10	1	0.163	10	3
	TAI040B	0.558	7	3	0.55	7	4	0.42	8	5	0.443	8	5	0.44	8	4
	TAI050B	0.253	10	1	0.285	10	1	0.261	10	1	0.281	10	0	0.228	10	1
	TAI060B	0.207	10	0	0.184	10	0	0.181	10	0	0.16	10	1	0.176	10	0
	TAI080B	0.461	8	0	0.337	10	0	0.477	8	0	0.461	9	0	0.594	9	0
	TAI100B	0.335	10	0	0.357	10	0	0.422	10	0	0.4	10	0	0.387	10	0
	TAI150B	0.71	9	0	0.639	10	0	0.622	10	0	0.585	10	0	0.684	10	0

Lentele nr. 6 Eksperimento II rezultatai (2 dalis)

		SPX			CX			DPX			OPX			OBX		
		Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis
II	TAI020B	0.09	10	8	0.085	10	8	0.101	10	8	0.09	10	8	0.091	10	8
	TAI025B	0.045	10	5	0.166	10	5	0.206	10	2	0.273	10	2	0.079	10	6
	TAI030B	0.551	8	2	0.514	8	2	0.69	8	0	0.37	9	1	0.56	8	2
	TAI035B	0.281	10	0	0.24	10	2	0.368	10	0	0.27	10	1	0.218	10	2
	TAI040B	0.068	10	5	0.464	8	3	0.411	8	1	0.352	8	4	0.515	8	4
	TAI050B	0.168	10	1	0.205	10	2	0.5	9	0	0.296	10	1	0.223	10	0
	TAI060B	0.163	10	0	0.183	10	1	0.33	10	0	0.176	10	0	0.201	10	0
	TAI080B	0.448	8	0	0.537	8	0	0.909	4	0	0.381	9	0	0.474	8	0
	TAI100B	0.471	10	0	0.458	10	0	0.644	10	0	0.429	10	0	0.37	10	0
	TAI150B	0.674	10	0	0.698	10	0	1.02	3	0	0.592	10	0	0.599	10	0

Lentele nr. 7 Eksperimento II rezultatai (3 dalis)

		MPX			COHX 1			COHX 2			COHX 3			COHX 4		
		Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis
II	TAI020B	0.09	10	8	0.045	10	9	0.045	10	9	0.056	10	9	0.056	10	9
	TAI025B	0.115	10	5	0.197	10	4	0.137	10	4	0.277	10	4	0.029	10	6
	TAI030B	0.346	9	0	0.381	9	0	0.338	9	1	0.384	9	1	0.259	10	1
	TAI035B	0.287	10	1	0.248	10	1	0.255	10	2	0.399	9	0	0.223	10	1
	TAI040B	0.245	9	6	0.542	7	4	0.036	10	6	0.263	9	6	0.423	8	5
	TAI050B	0.298	9	1	0.282	10	0	0.133	10	2	0.276	10	0	0.195	10	0
	TAI060B	0.137	10	0	0.212	10	0	0.144	10	0	0.198	10	0	0.174	10	1
	TAI080B	0.591	7	0	0.465	8	0	0.435	8	0	0.58	10	0	0.56	10	0
	TAI100B	0.305	10	0	0.399	10	0	0.362	10	0	0.41	10	0	0.407	10	0
	TAI150B	0.646	10	0	0.647	10	0	0.582	10	0	0.737	10	0	0.738	10	0

Lentele nr. 8 Eksperimento III rezultatai (1 dalis)

		ULX			RULX			RX			BX			UPMX		
		Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis
III	TAI020A	0.183	10	4	0.238	10	4	0.216	10	4	0.216	10	4	0.238	10	4
	TAI025A	0.185	10	6	0.154	10	6	0.191	10	5	0.19	10	5	0.117	10	7
	TAI030A	0.112	10	7	0.079	10	8	0.097	10	7	0.091	10	7	0.111	10	7
	TAI035A	0.29	10	2	0.348	10	1	0.388	10	0	0.175	10	5	0.284	10	2
	TAI040A	0.483	10	0	0.465	10	0	0.474	10	0	0.454	10	0	0.455	10	0
	TAI050A	0.678	10	0	0.628	10	0	0.609	10	0	0.638	10	0	0.565	10	0
	TAI060A	0.655	10	0	0.646	10	0	0.629	10	0	0.608	10	0	0.623	10	0
	TAI080A	0.274	10	0	0.267	10	0	0.295	10	0	0.219	10	1	0.265	10	0
	TAI100A	0.224	10	0	0.219	10	0	0.203	10	0	0.192	10	0	0.191	10	0

Lentele nr. 9 Eksperimento III rezultatai (2 dalis)

		SPX			CX			DPX			OPX			OBX		
		Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis
III	TAI020A	0.207	10	5	0.268	10	3	0.221	10	4	0.202	10	5	0.268	10	3
	TAI025A	0.151	10	6	0.158	10	6	0.231	10	4	0.19	10	5	0.159	10	6
	TAI030A	0.091	10	7	0.04	10	9	0.074	10	7	0.12	10	6	0.112	10	7
	TAI035A	0.283	10	2	0.312	10	1	0.433	10	0	0.209	10	4	0.324	10	0
	TAI040A	0.444	10	0	0.424	10	0	0.503	10	0	0.441	10	0	0.425	10	0
	TAI050A	0.585	10	0	0.627	10	0	0.693	10	0	0.602	10	0	0.579	10	0
	TAI060A	0.609	10	0	0.619	10	0	0.689	10	0	0.596	10	0	0.613	10	0
	TAI080A	0.24	10	0	0.251	10	0	0.33	10	0	0.268	10	0	0.26	10	0
	TAI100A	0.142	10	0	0.165	10	0	0.23	10	0	0.178	10	0	0.219	10	0

Lentele nr. 10 Eksperimento III rezultatai (3 dalis)

		MPX			COHX 1			COHX 2			COHX 3			COHX 4		
		Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis
III	TAI020A	0.246	10	3	0.263	10	3	0.229	10	3	0.113	10	7	0.077	10	8
	TAI025A	0.15	10	6	0.154	10	6	0.146	10	6	0.037	10	9	0.052	10	8
	TAI030A	0.035	10	8	0.075	10	7	0.097	10	7	0.002	10	9	0.002	10	9
	TAI035A	0.193	10	4	0.284	10	2	0.169	10	5	0.13	10	6	0.144	10	5
	TAI040A	0.383	10	0	0.461	10	0	0.383	10	0	0.256	10	0	0.261	10	0
	TAI050A	0.561	10	0	0.686	10	0	0.601	10	0	0.522	10	0	0.501	10	0
	TAI060A	0.586	10	0	0.642	10	0	0.544	10	0	0.556	10	0	0.528	10	0
	TAI080A	0.218	10	0	0.248	10	0	0.261	10	0	0.184	10	0	0.168	10	0
	TAI100A	0.121	10	0	0.144	10	0	0.158	10	0	0.119	10	0	0.105	10	1

Lentele nr. 11 Eksperimento IV rezultatai (1 dalis)

		ULX			RULX			RX			BX			UPMX		
		Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis
IV	TAI020B	0	10	10	0	10	10	0	10	10	0	10	10	0	10	10
	TAI025B	0	10	10	0	10	10	0	10	10	0	10	10	0	10	10
	TAI030B	0	10	9	0	10	9	0.018	10	7	0.001	10	8	0.001	10	6
	TAI035B	0.048	10	7	0.049	10	7	0.014	10	9	0.049	10	7	0.019	10	9
	TAI040B	0	10	10	0	10	10	0	10	10	0	10	10	0	10	10
	TAI050B	0.003	10	9	0.035	10	6	0.003	10	9	0.001	10	9	0.006	10	8
	TAI060B	0.011	10	7	0.024	10	5	0.004	10	8	0.012	10	6	0.012	10	4
	TAI080B	0.049	10	3	0.004	10	7	0.067	10	3	0.012	10	4	0.119	10	6
	TAI100B	0.074	10	5	0.104	10	3	0.091	10	2	0.115	10	1	0.117	10	1
	TAI150B	0.329	10	0	0.305	10	0	0.091	10	0	0.359	10	0	0.41	10	0

Lentele nr. 12 Eksperimento IV rezultatai (2 dalis)

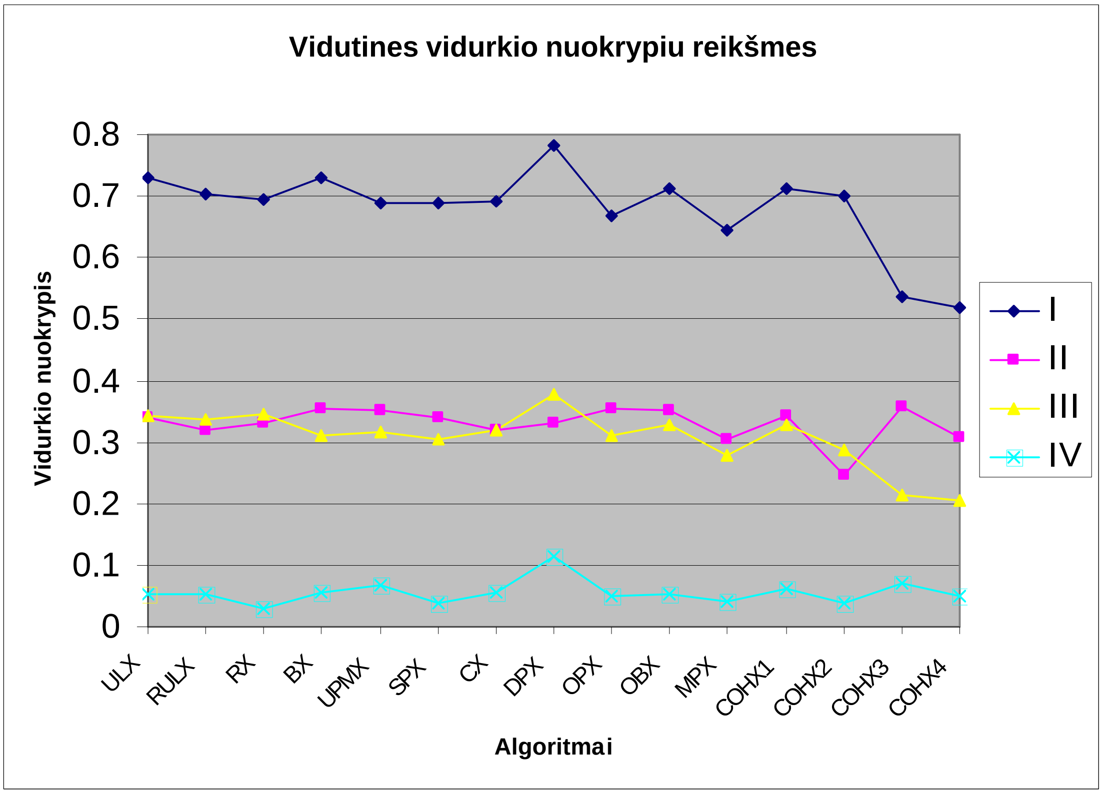
		SPX			CX			DPX			OPX			OBX		
		Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis
IV	TAI020B	0	10	10	0	10	10	0	10	10	0	10	10	0	10	10
	TAI025B	0	10	10	0	10	10	0.007	10	9	0	10	10	0	10	10
	TAI030B	0.001	10	6	0.017	10	7	0.019	10	4	0.001	10	7	0	10	9
	TAI035B	0.029	10	8	0.042	10	6	0.047	10	8	0.062	10	7	0.037	10	8
	TAI040B	0	10	10	0	10	10	0.045	10	9	0	10	10	0	10	10
	TAI050B	0.012	10	9	0.002	10	7	0.088	10	2	0	10	10	0	10	9
	TAI060B	0.002	10	8	0.007	10	7	0.025	10	2	0.002	10	8	0.015	10	3
	TAI080B	0.017	10	3	0.013	10	3	0.173	10	2	0.045	10	2	0.017	10	3
	TAI100B	0.091	10	2	0.108	10	2	0.145	10	1	0.113	10	1	0.114	10	2
	TAI150B	0.232	10	0	0.36	10	0	0.608	10	0	0.261	10	0	0.338	10	0

Lentele nr. 13 Eksperimento IV rezultatai (3 dalis)

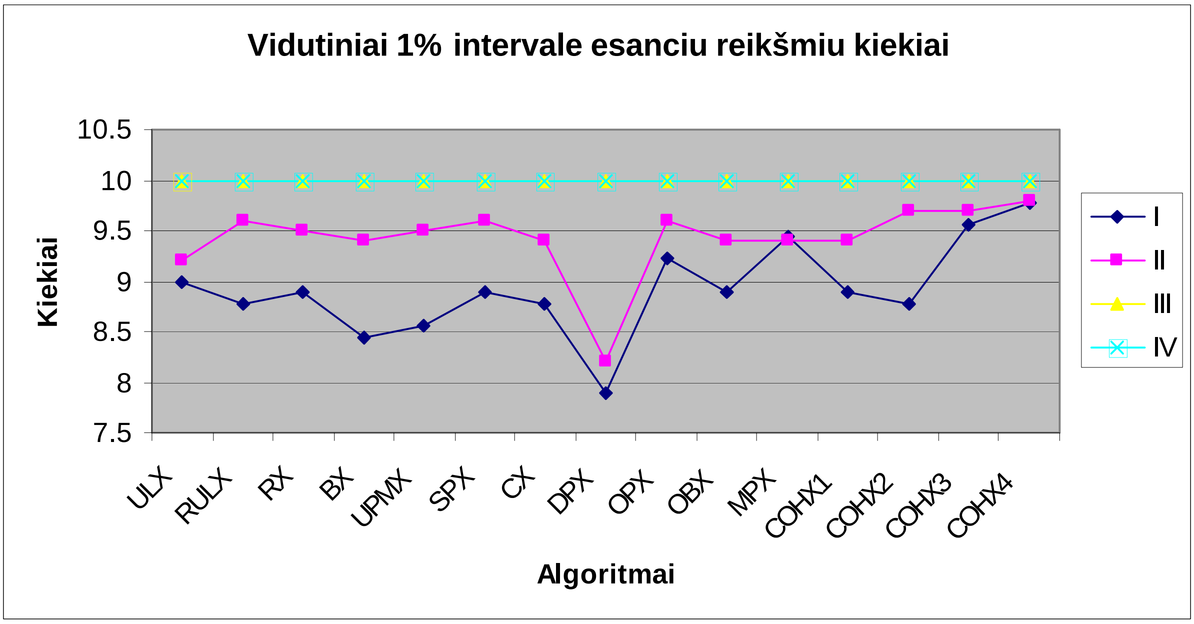
		MPX			COHX 1			COHX 2			COHX 3			COHX 4		
		Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis	Vidurkio nuokrypis	1 % intervale	Geriausiu kiekis
IV	TAI020B	0	10	10	0	10	10	0	10	10	0	10	10	0	10	10
	TAI025B	0	10	10	0	10	10	0	10	10	0.007	10	9	0	10	10
	TAI030B	0	10	9	0	10	9	0	10	9	0.014	10	8	0	10	9
	TAI035B	0	10	10	0.048	10	7	0.024	10	9	0.031	10	8	0.052	10	7
	TAI040B	0	10	10	0	10	10	0	10	10	0	10	10	0	10	10
	TAI050B	0.001	10	9	0.001	10	9	0.006	10	8	0.004	10	7	0	10	10
	TAI060B	0.012	10	6	0.005	10	5	0	10	8	0.029	10	4	0.005	10	9
	TAI080B	0.003	10	7	0.1	10	3	0.016	10	1	0.16	10	2	0.059	10	4
	TAI100B	0.07	10	5	0.116	10	0	0.081	10	2	0.114	10	0	0.098	10	3
	TAI150B	0.311	10	0	0.335	10	0	0.255	10	0	0.35	10	0	0.277	10	0

Pastaba:
ULX – tolygiojo pasiskirstymo krosoveris
RULX – atsitiktinis ULX
RX – koreguojantis krosoceris
BX – blokinis krosoveris
UPMX – tolygusis dalinio planavimo krosoveris
SPX – poriniu sukeitimu krosoveris
CX – ciklinis krosoveris
DPX – išlaikantis atstuma krosoveris
OPX – vieno taško krosoveris
OBX – tvarka paremtas krosoveris
MPX – keleto tevu krosoveris
COHX1 – surišantis krosoveris, nenaudojantis bitu kaukes ir nevertinantis tevu „gerumo“
COHX2 - surišantis krosoveris, nenaudojantis bitu kaukes ir vertinantis tevu „geruma“
COHX3 - surišantis krosoveris, naudojantis bitu kauke ir nevertinantis tevu „gerumo“
COHX4 - surišantis krosoveris, naudojantis bitu kauke ir vertinantis tevu „geruma“

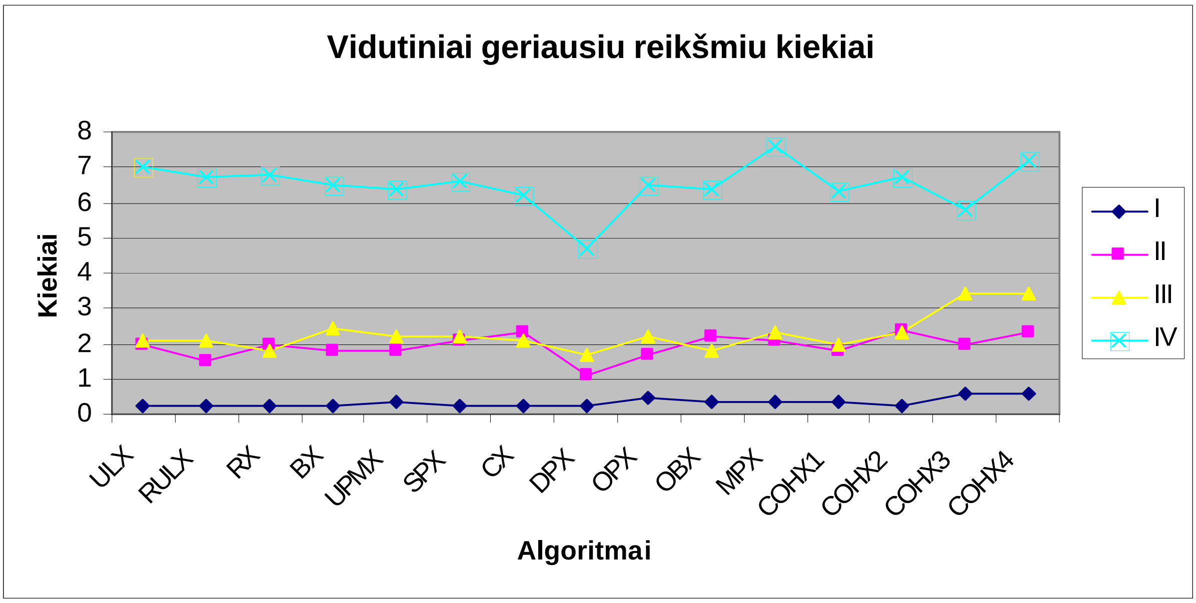
Kadangi gautas labai didelis rezultatu kiekis, grafiškai pateikiamos tik vidutines rezultatu reikšmes.



21 pav. Vidutiniu vidurkiu nuokrypiu grafikas



22 pav. Vidutiniu 1% intervale esanciu reikšmiu kiekiu grafikas



23 pav. Vidutiniu geriausiu reikšmiu kiekiu grafikas

Iš rezultatu matome, jog algoritmas kur kas lengviau „susidoroja“ su „realaus pasaulio“ duomenis imituojanciais pavyzdžiais. Padidinus generaciju kieki, rezultatai taip pat gaunami geresni.

Geriausios vidurkio nuokrypio reikšmes gaunamos naudojant COHX3 ir COHX4 kryžminimo operacijas, pastarieji aprašyti kaip modifikuoti surišantieji krosoveriai. Skirtumas tarp ju tik tas, jog COHX3 algoritme tevu „gerumas“ nevertinamas, kai COHX4 algoritme pirmo tevo tikslo funkcijos reikšme turi buti geresne. Iš 21 pav. akivaizdu, jog toks tevu ivertinimas pagerina HGA rezultatus.

Pagal vidutiniu 1% intervale esanciu reikšmiu kiekiu grafika, galima analizuoti tik I ir II eksperimentu rezultatus. III ir IV eksperimentu metu, visi algoritmai parode vienodai gerus rezultatus. Pirmu eksperimentu metu COHX3 ir COHX4 vel parode geriausius rezultatus. Akivaizdu, jog ir pagal 1% intervale esanciu reikšmiu kieki, prieš atliekant kryžminimo operacija, verta ivertinti tevu „geruma“.

Analizuojant gautus geriausiu reikšmiu kiekius, pastebimas vel COHX4 algoritmo pranašumas. Taciau IV eksperimento metu dideli pagerejima pademonstravo MPX algoritmas. Nors kitu eksperimentu metu, šio algoritmo vidutiniai rezultatai neišsiskyre iš kitu algoritmu.

6. Išvados

1. Kvadratinio paskirstymo uždavinys, kaip ir daugelis kitu kombinatorinio optimizavimo uždaviniu, del jo sudetingumo išsprendžiamas tiksliai tik tuomet, kai jo apimtis labai ribota. Todel šiam uždaviniui spresti taikomi euristiniai algoritmai. Vienas iš tokiu algoritmu – hibridinis genetinis algoritmas (HGA), kurio viena iš pagrindiniu daliu yra kryžminimo operatorius.

2. Šiame darbe realizuotas sprendiniu kryžminimo algoritmas pagal aprašyma, pateikta Z. Drezner straipsnyje "A New Genetic Algorithm for the Quadratic Assignment Problem" [21]. Taip pat realizuota keleta šio algoritmo modifikaciju, kurio leis ivertinti Z. Drezner pasiulyto algoritmo pranašumus arba trukumus.

3. Algoritmai buvo išbandyti su ivairiu tipu kvadratinio paskirstymo uždavinio testiniais pavyzdžiais (duomenimis) iš kvadratinio paskirstymo uždavinio duomenu bibliotekos QAPLIB. Rezultatai buvo palyginti su jau kitais gerai žinomais kryžminimo algoritmu rezultatais.

4. Realizuoti surišancio „krosoverio" algoritmai pademonstravo gerus rezultatus.

5. Net ir esant pasiektiems geriems rezultatams, tikslinga toliau ieškoti kitu patobulinimo budu, strategiju ir ideju.

6. Kiti galimi hibridinio genetinio algoritmo patobulinimai galetu buti susije su kitu genetiniu operatoriu, pvz. lokalios paieškos modifikavimu. Galetu buti išbandyti ir kiti specialus mechanizmai, pvz., imigravimas, konkuruojancios populiacijos ir t.t.

7. Patobulintus hibridinius genetinius algoritmus, gaunancius gerus rezultatus sprendžiant kvadratinio pasiskirstymo uždavini, butu galima išbandyti kitiems aktualiems kombinatorinio optimizavimo uždaviniams, pvz., komivojažieriaus uždaviniui ir pan.

7. Literatura

[1] **Ahuja R. K., Orlin J. B., Tiwari A.,** 2000. A greedy genetic algorithm for the quadratic assignment problem. *Computers and Operations Research* 27, 917–934.

[2] **Anily S., Federgruen A.** Simulated annealing methods with general acceptance probability. *Journal of Applied Probability*, 1987, vol.24, 657-667.

[3] **Boelte A., Thonemann U. W.** Optimizing simulated annealing schedules with genetic programming// *European Journal of Operational Research, 1996, p. 92,* 402–416.

[4] **Boese K.D., Kahng A.B., Muddu S.** A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 1994, *Vol*.16, 101–113.

[5] **Burkard R.E., Çela E., Pardalos P.M., Pitsoulis L..** The quadratic assignment problem. In D.Z.Du, P.M.Pardalos (eds.), *Handbook of Combinatorial Optimization*, vol.3, Dordrecht: Kluwer, 1998, 241-337.

[6] **Burkard R. E., Karisch S., Rendl F.** QAPLIB – a quadratic assignment problem library. *Journal of Global Optimization*, 1997, p. 10, 391–403.

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

[8] **Davis L.** Order-based genetic algorithms and the graph coloring problem. In L.Davis (ed.), *Handbook of Genetic Algorithms*, Van Nostrand, New York, 1991, 72–90.

[9] **Dorigo M., Di Caro G.** The ant colony optimization metaheuristic. In D.Corne, M.Dorigo, F.Glover (eds.), *New Ideas in Optimization*, New York: McGraw-Hill, 1999, 11-32.

[10] **Drezner Z.** A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing*, 2003, *Vol*.15, 320–330.

[11] **Elshafei A. N.** Hospital layout as a quadratic assignment problem. *Operations Research Quarterly*, 1977, *Vol*.28, p. 167–179.

[12] **Fleurent C., Ferland J.A.,** 1993. Genetic hybrids 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*

[13] **Fleurent C., Glover F.** Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing*, 1999, *Vol*.11, 198–204.

[14] **Glover F.** Genetic algorithms and scatter search: unsuspected potential. *Statistics and Computing*, 1994, Vol.4, 131–140.

[15] **Glover F.** Tabu search: part I. *ORSA Journal on Computing*, 1989, Vol.1, p. 190–206.

[16] **Glover F.** Tabu search: part II. *ORSA Journal on Computing,* 1990, Vol.2, p. 4–32.

[17] **Goldberg DE.** Genetic algorithms in search, optimization and machine learning, *Addison-Wesley, Reading, MA,* 1989

[18] **Goldberg DE, Lingle R.** Alleles, loci, and the traveling salesman problem. In J.J.Grefenstette (ed.), *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, Lowrence Erlbaum, Hillsdale, 1985, 154–159.

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

[20] **Koopmans T., Beckmann M.** Assignment problems and the location of economic activities. *Econometrica*, 1957, Vol.25, p. 53–76.

[21] **Lim M. H., Yuan Y., Omatu S.,** Efficient Genetic Algorithms Using Simple Genes Exchange Local Search Policy for the Qaudratic Assignment Problem. *Kluwer Academic Publishers,* 2000

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

[23] **Merz P., Freisleben B.,** 1997a. A genetic local search approach to the quadratic assignment problem.

[24] **Migkikh V.V., Topchy A.A., Kureichik V.M., Tetelbaum A.Y..** Combined genetic and local search algorithm for the quadratic assignment problem. Proceedings of the First International Conference on Evolutionary Computation and its Applications (EVCA'96), Presidium of the Russian Academy of Sciences, Moscow, 1996, 335–341.

[25] **Misevicius A., Rubliauskas D.** Performance of hybrid genetic algorithm for the grey pattern problem. *Information Technology and Control*, 2005, *Vol*.34, *No*.1, 15–24.

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

[27] **Radcliffe N., Surry P.** Fitness variance of formae and performance prediction. In L.D.Whitley, M.Vose (eds.), *Proceedings of the Third Workshop on Foundations of Genetic Algorithms,* Morgan Kaufmann, San Francisco, 1994, 51–72.

[28] **M.G.C.Resende, C.C.Ribeiro.** Greedy randomized adaptive search procedures. In F.Glover, G.Kochenberger (eds.), *Handbook of Metaheuristics,* Norwell: Kluwer, 2002, 218-249.

[29] **Sahni S., Gonzalez T.** P-complete approximation problems// *Journal of ACM*, 1976, p. 23, 555–565.

[30] **Skorin-Kapov J.** Tabu search applied to the quadratic assignment problem// *ORSA Journal on Computing*, 1990, p. 2, 33–45.

[31] **Tate D. M., Smith A. E.,** A genetic approach to the quadratic assignment problem. *Computers and Operations Research*, 1995, *Vol*.1, 73–83.

8. Paveiksleliai

9. Summary

Hybrid Genetic Algorithm and its modifications for the Qaudratic Assignment Problem

Genetic algorithms (GA) are among the widely used in various areas of computer science, including optimization problems. Genetic algorithms (GA) are based on the biological process of natural selection. Many simulations have demonstrated the efficiency of GAs on different optimization problems, among them, bin-packing, qaudratic assignment problem, graph partitioning, job-shop scheduling problem, set covering problem, traveling salesman problem, vehicle routing.

The quadratic assignment problem (QAP) belong to the class of NP-hard combinatorial optimization problems. The QAP can be formulated in many ways. We consider a variant of the many equivalent forms as follows. Given a QAP of size n described by two n x n cost matrices $\boldsymbol{A} = [a_{ij}]$ and $\boldsymbol{B} = [b_{ij}]$, the goal is to find a permutation $\boldsymbol{p}$ of the set $\boldsymbol{P} = \{1, 2, 3, \ldots, n\}$, which minimizes the objective function $z(\boldsymbol{p})$.

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

One of the main operators in GA is a crossover (i.e. solution recombination). This operator plays a very important role by constructing competitive genetic algorithms (GAs). In this work, we investigate several crossover operators for the QAP, among them, ULX (uniform like crossover), SPX (swap path crossover), OPX (one point crossover), COHX (cohesive crossover), MPX (multiple parent crossover) and others. Comparison of these crossover operators was performed. The results show high efficiency of the cohesive crossover.

10. Priedas

Surišancio krosoverio, naudojancio bitu kauke, išeities tekstas pateiktas žemiau:

```
unit cohesive;
interface
{$I header.dcl}

   const CMatrixMax = 50;

   Type TBitMatrix = array[1..CMatrixMax, 1..CMatrixMax] of boolean;
        TDistMatrix = array[1..CMatrixMax, 1..CMatrixMax] of integer;

   procedure cohesive_crossover(p_1, p_2: fixed_int_array;var o: fixed_int_array);

   procedure cohesive_merge(dydis:integer; poz:integer;s1,s2:fixed_int_array;
                            var s3:fixed_int_array);

   function exist(reiksme, mas_dydis: integer; mas: fixed_int_array;
                  var poz: integer):integer;
   function add2arr(reiksme: integer;
                    var mas_dydis: integer; var mas: fixed_int_array):integer;
   function get_pozic(dydis_x, poz_x, poz_y: integer):integer;
   procedure get_koord(dydis_x, poz: integer; var x: integer; var y: integer);
   procedure get_size(ilgis: integer; var x_size: integer; var y_size: integer);
   function med_dist(x1, y1, x2, y2: integer): integer;

   procedure get_koord_new(SSize, nx, ny, poz: integer;
                           var x: integer; var y: integer);
   function get_pozic_new(SSize, nx, ny, poz_x, poz_y: integer):integer;

implementation

uses objectiv, protocol, utils;
{****}
   Procedure PrintBitMatrix(mat:TBitMatrix; size:integer);
      var x, y: integer;
   begin
      writeln;
      for y:=1 to size do begin
         for x:=1 to size do if mat[x,y] then write('1':2)
                                         else write('0':2);
```

```
      writeln;
    end;
  end;

  Procedure PrintDistMatrix(mat:TDistMatrix; size, maxDist:integer);
    var x, y: integer;
  begin
    writeln;
    for y:=1 to size do begin
      for x:=1 to size do if mat[x,y]>=0 then write(mat[x,y]:3)
                                           else write('  .');
      writeln;
    end;
    Writeln('Max. Distance ',maxDist);
  end;

  procedure PrintArray(s:fixed_int_array; dydis, SSize, ny:integer);
   var ix, iy, i: integer;
  begin
    writeln;
    for i:=1 to dydis do begin
      write(s[i]:3);
      if (SSize - ny) * SSize >= i then begin
        if i mod SSize = 0 then writeln
      end else
        if (i - (SSize - ny)*SSize) mod (SSize-1) = 0 then writeln
    end;
    writeln;
  end;
{***************************************************************************}
{**** Medianinis atstumas tarp dvieju tasku ********************************}
{***************************************************************************}
  function med_dist(x1, y1, x2, y2: integer): integer;
  begin
    med_dist := abs( x1 - x2 ) +  abs( y1 - y2 );
  end;

{***************************************************************************}
{**** Randamas matricos dydis, kurios krastiniu suma maziausia *************}
{***************************************************************************}
  procedure get_size(ilgis: integer; var x_size: integer; var y_size: integer);
    var i, x, y: integer;
  begin
    x_size := ilgis;
    y_size := 1;
```

```
      for i:=1 to (round(sqrt(ilgis))) + 1 do
         if ilgis mod i = 0 then
            if x_size + y_size > i + ilgis div i then begin
               x_size := ilgis div i;
               y_size := i;
            end;
   end;

{**************************************************************************}
{**** Konvertuoja adresa is 1d i 2d (apkarpytoje matricoje) ***************}
{**************************************************************************}
procedure  get_koord_new(SSize,  nx,  ny,  poz:  integer;  var  x:  integer;  var  y:
integer);
begin
   if SSize * (SSize - ny) >= poz
      then get_Koord(SSize,poz,x,y)
      else begin
              get_Koord(SSize-1,poz - SSize * (SSize - ny),x,y);
              y := y + (SSize - ny);
      end;
end;

{**************************************************************************}
{**** Konvertuoja adresa is 1d i 2d ***************************************}
{**************************************************************************}
procedure get_koord(dydis_x, poz: integer; var x: integer; var y: integer);
begin
   y:= (poz-1) div dydis_x+1;
   x:= poz - ((y-1) * dydis_x);
end;

{**************************************************************************}
{**** Konvertuoja adresa is 1d i 2d (apkarpytoje matricoje) ***************}
{**************************************************************************}
   function get_pozic_new(SSize, nx, ny, poz_x, poz_y: integer):integer;
      var pag: integer;
   begin
      if poz_y <= (SSize - ny) then begin
         pag := get_pozic(SSize,poz_x,poz_y);
      end else begin
         pag := get_pozic(SSize-1,poz_x,poz_y) + (SSize - ny);
      end;
      get_pozic_new := pag;
   end;
```

```
{**************************************************************************}
{**** Konvertuoja adresa is 2d i 1d ***************************************}
{**************************************************************************}
   function get_pozic(dydis_x, poz_x, poz_y: integer):integer;
      var pag:integer;
   begin
      pag := dydis_x * (poz_y - 1) + poz_x;
      get_pozic := pag;
   end;

{**************************************************************************}
{**** Tikrina ar reiksme unikali masyve ***********************************}
{**************************************************************************}
   function   exist(reiksme,   mas_dydis:   integer;   mas:   fixed_int_array;   var
poz:integer):integer;
      var pag: integer;
          i: integer;

          kp, dp, vp: integer;  //Kaire puse, Desine puse, Vidurine puse
   begin
      pag := 0;
      poz := 1;
      if mas_dydis>0 then begin
         kp := 1;                //Kaire masyvo Reiksme
         dp := mas_dydis;        //Desine masyvo Reiksme
         vp := (mas_dydis+1) div 2;  //Vidurine masyvo Reiksme
         while (vp<>kp) and (vp<>dp)
               and (mas[kp] <> reiksme)
               and (mas[vp] <> reiksme)
               and (mas[dp] <> reiksme) do begin

            if reiksme < mas[vp] then begin
               dp := vp;
               vp := kp + ((dp - kp+1) div 2);

            end else begin
               kp := vp;
               vp := kp + ((dp - kp+1) div 2);

            end;

         end;
         if reiksme = mas[kp] then pag := kp;
         if reiksme = mas[dp] then pag := dp;
         if reiksme = mas[vp] then pag := vp;
```

```
        //Jeigu nerasta reiksme tuomet grazinama kur turetu buti
        poz := pag;
        if pag = 0 then begin
           if reiksme < mas[kp] then poz := kp
           else if reiksme < mas[dp] then poz := dp
                else poz:= dp + 1;
        end;
     end;
     exist := pag;
  end;

{*****************************************************************************}
{**** Reiksme patalpinama i masyva issaugant masyve didejimo tvarka ********}
{*****************************************************************************}
  function  add2arr(reiksme:  integer;  var  mas_dydis:  integer;  var  mas:
fixed_int_array):integer;
     var i, k, poz, kur: integer;
  begin
     kur := exist( reiksme, mas_dydis, mas, poz);

     if kur = 0 then begin
        if poz>mas_dydis then begin

           mas_dydis := poz;
           mas[poz] := reiksme;

        end else begin

           for i := mas_dydis downto poz do mas[i+1]:= mas[i];
           mas[poz] := reiksme;
           mas_dydis := mas_dydis + 1;

        end;
        add2arr := poz;
     end else add2arr := kur;
  end;

{*****************************************************************************}
{**** Sukryzmina tevus ir gaunamas "vaikas" *******************************}
{*****************************************************************************}
{mx, my - matricos dydis
 fx, fy - kordinates nuo kurios formuojamas distanciju matrica
 median - mediana}
  procedure merge(Dydis, SSize, nx, ny: integer;
                   MaxDist: integer;
```

```
              MMat: TBitMatrix;
              DMat: TDistMatrix;
              s1, s2: fixed_int_array; var s3: fixed_int_array);

   var ix, iy, ip, i, ii, truk_ilg: integer;
       truk: fixed_int_array; {Poziciju ir trukstanciu reiksmiu masyvai}
       yra: boolean;
       pag: integer;
       yra_vidur: boolean;
       domin,panaudoti,pozicijos, vidurio_poz, pvid_poz: fixed_int_array;
       domin_ilg, panaudoti_ilg, pozicijos_ilg,
       vidurio_poz_ilg, pvid_poz_ilg: integer;
       median: real;
begin
   median := MaxDist / 2 + 0.001;
   if MaxDist mod 2 = 0 then yra_vidur:=true
                        else yra_vidur:=false;

   //Sulipinami tevai
   domin_ilg := 0;
   pozicijos_ilg := 0;
   vidurio_poz_ilg := 0;
   for iy:=1 to SSize do begin
      for ix:=1 to SSize do begin

         if MMat[ix, iy] then begin
            //Jeigu atstumas kaip tik "vidurinis" tuomet uzsaugome ji atskirai
            ip := get_pozic_new(SSize, nx, ny, ix, iy);
            if (yra_vidur) and (DMat[ix,iy]=round(median)) then begin
              //Issaugomos pozicijos kuriose galetu buti is 1-mo tevo
               add2arr( ip, vidurio_poz_ilg, vidurio_poz );
            end;

            //Maziau uz Mediana priskiriame is pirmo tevo, daugiau is antro
            if DMat[ix,iy]< median then begin
               s3[ip] := s1[ip]; {is pirmo tevo - geresnio sprendinio}
               add2arr( s3[ip], domin_ilg, domin );
            end
               else begin
                      s3[ip] := s2[ip];{is antro tevo}
                    end;
         end;

      end;
   end;
```

```
//ismaisomos vidurines reiksmes
pvid_poz_ilg := 0;
if yra_vidur then begin
   for ix:=1 to vidurio_poz_ilg do begin
      i := random(vidurio_poz_ilg)+1;
      ii := vidurio_poz[ix];
      vidurio_poz[ix] := vidurio_poz[i];
      vidurio_poz[i] := ii;
   end;
   //Puse ju surasomos i rezultato sprendini
   for ix:=1 to vidurio_poz_ilg div 2 do begin
      s3[ vidurio_poz[ix] ] := s1[ vidurio_poz[ix] ];
      add2arr( vidurio_poz[ix], pvid_poz_ilg, pvid_poz );
      add2arr( s3[ vidurio_poz[ix] ], domin_ilg, domin );
   end;
end;


//Isrenkamos pasikartojancios reiksmes
//ir issaugoma pozicijos kuriose tos reiksmes kartojasi
panaudoti := domin;
panaudoti_ilg := domin_ilg;
for iy:=1 to SSize do begin
   for ix:=1 to SSize do begin
      //i := med_dist(fx, fy, ix, iy);
      if MMat[ix,iy] then begin
         ip := get_pozic_new(SSize, nx, ny, ix, iy);
         if (DMat[ix,iy] > median) and
            (exist(ip, pvid_poz_ilg, pvid_poz, pag)=0) then begin

            if exist(s3[ip], domin_ilg, domin, pag) > 0 then begin
               add2arr( ip, pozicijos_ilg, pozicijos );
               s3[ip] := 0;
            end else add2arr( s3[ip], panaudoti_ilg, panaudoti );
         end;
      end;

   end;
end;

//Isrenkamos nepanaudotos reiksmes
truk_ilg:=0;

for i:=1 to SSize * SSize do begin
```

```
        if (i<=dydis) then begin
            get_koord_new(SSize, nx, ny, i, ix, iy);

            if exist(i, panaudoti_ilg, panaudoti, pag) = 0 then begin
               truk_ilg := truk_ilg + 1;
               truk[truk_ilg]:=i;
            end;

        end;

      end;


      //Atsitiktinai ismaisomos nepanaudotos reiksmes
      for i:=1 to truk_ilg do begin
         ii := random(truk_ilg) + 1;
         ip := truk[i];
         truk[i] := truk[ii];
         truk[ii] := ip;
      end;


      //Atsitiktinai uzpildomos tuscios pozicijos
      for i:=1 to truk_ilg do begin
         s3[ pozicijos[i] ] := truk[i];
      end;

   end;

{****************************************************************************}
{**** Cohesive merge *******************************************************}
{****************************************************************************}
procedure cohesive_merge(dydis:integer; poz:integer;s1,s2:fixed_int_array;
                        var s3:fixed_int_array);
      var i, l: integer;
          nx, ny: integer;

          median: real;
          fx, fy: integer;
          ix, iy, ip: integer;
          SSize: integer;
          BitMat: TBitMatrix;
          DistMat: TDistMatrix;
          MaxDist: integer;
          pp: fixed_int_array;
```

```
begin

   SSize := round(sqrt(dydis));
   if sqr( SSize ) < dydis then begin
      SSize := SSize + 1;
   end;



   //Formuojama MASKES matrica
   for iy:=1 to SSize do
      for ix:=1 to SSize do begin
         BitMat[ix, iy] := true;
      end;
   //Suskaiciuojama nuliu kiekis maskes krastinese
   if (SSize * SSize - dydis) mod 2 = 0 then begin
      nx := round((SSize * SSize - dydis + 0.001) / 2);
      ny := round((SSize * SSize - dydis + 0.001) / 2) + 1;
   end else begin
      nx := round( (SSize * SSize - dydis + 0.001) / 2 );
      ny := round( (SSize * SSize - dydis + 0.001) / 2 );
   end;

   if (nx = 0) or (ny = 0) then begin
      nx := 0;
      ny := 0;
   end;

   for i:=1 to nx do  BitMat[SSize - (i-1),SSize] := false;
   for i:=1 to ny do  BitMat[SSize,SSize - (i-1)] := false;

   //Formuojama DISTANCE matrica
   MaxDist := -1;

   get_koord_new(SSize, nx, ny, poz, fx, fy);

   for iy:=1 to SSize do
      for ix :=1 to SSize do begin
         if BitMat[ix,iy] then
            begin
               DistMat[ix, iy] := med_dist(ix, iy, fx, fy);
               if DistMat[ix,iy] > MaxDist then MaxDist := DistMat[ix,iy];
            end
            else DistMat[ix, iy] := -1;
      end;
```

```
      merge(dydis, SSize, nx, ny, MaxDist, BitMat, DistMat, s1, s2, s3);
end;

procedure cohesive_crossover(p_1, p_2: fixed_int_array; var o: fixed_int_array);
   var l_sp, l_sp_min: longint;    //Laikinas sprendinys isrenkant geriausia
       o_sp: fixed_int_array;      //Laikinas perstatymas isrenkant geriausia
       Fp1, Fp2: longint;          //Tevu tikslo func. reiksmes
       i: integer;
begin

     prot_lf_text('p_1:');
     for i:=1 to N do prot_text(' '+numtostr(p_1[i],2,0));
     prot_lf_text('p_2:');
     for i:=1 to N do prot_text(' '+numtostr(p_2[i],2,0));


   l_sp := INFINITY;              //Maxsimali reiksme
   l_sp_min := INFINITY;
   for i := 1 to N do begin
      cohesive_merge(N, i, p_1, p_2, o_sp);
      l_sp := objective_function_value( o_sp );
      if l_sp < l_sp_min then begin
                               l_sp_min := l_sp;
                               move(o_sp, o, N_Intsize);
                            end;
   end;

     prot_lf_text('o  :');
     for i:=1 to N do prot_text(' '+numtostr(o[i],2,0));
end;

begin
end.
```