VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Euristiniai algoritmai grafų palyginimui

(Heuristic Algorithms for Comparing Graphs)

Magistro baigiamasis darbas

Atliko:	Karolis Žukauskas	(parašas)
Darbo vadovas:	Lekt. Irmantas Radavičius	(parašas)
Recenzentas:	Asist., Dr. Gintaras Skersys	(parašas)

Vilnius – 2021

Santrauka

Šiame darbe atliekamas empirinis grafų redagavimo atstumo (toliau GRA) tyrimas siekiant paspartinti bei patikslinti tiksliausius esamus euristinius algoritmus. Pirmoje darbo dalyje apibrėžiamas grafų redagavimo atstumo uždavinys bei detaliai aprašomos trys euristinių algoritmų klasės. Antroje darbo dalyje yra aprašomi eksperimentuose naudoti grafų rinkiniai ir pateikiamas empirinis grafų redagavimo atstumo tyrimas. Pagrindinis šiame darbe analizuojamas algoritmas yra lokalia paieška paremtas *IPFP* [BDB+18] algoritmas, kuris šiai dienai išgauna patį tiksliausią viršutinį GRA rėžį [BBG+20]. Siekiant paspartinti algoritmą *IPFP* bei kartu ir patikslinti šio algoritmo išgaunamus rėžius, reikia parinkti tinkamą algoritmą lokalios paieškos pradiniam sprendiniui surasti. Šiam darbui atlikti geriausiai tinka greiti, tačiau ne itin tikslūs tiesinių sumų priskyrimo su klaidų taisymu klasės (toliau TSPKT-GRA) algoritmai. Šiame darbe parodoma, kad algoritmo parinkimas pradiniam sprendiniui rasti yra sudėtingas uždavinys, kurį veikia ne tik grafo viršūnių ar briaunų skaičius, bet ir grafo struktūra. Darbe pristatomos kelios TSPKT-GRA klasės algoritmų modifikacijos: *Star-5*, *Branch-Const-2* ir *Branch-Const-4*, kurios parodė gerus rezultatus GRA skaičiavime. Norint greito, tačiau ne tokio tikslaus, GRA skaičiavimo, reikėtų rinktis šiame darbe pristatytą algoritmą *Branch-Const-4*. Norint išgauti tikslesnius GRA rėžius reiktų taikyti keliomis eilėmis lėtesnį, tačiau žymiai tikslesnį algoritmą *IPFP*, kurio pradiniam sprendiniui surasti yra taikomas vienas iš TSPKT-GRA algoritmų. Nedidelio dydžio retiems grafams reiktų rinktis algoritmą *Refine(BP)* arba *Refine(Branch-Fast)*, vidutinio dydžio grafams reiktų rinktis algoritmą *Star-5*, dideliuose arba tankiuose grafuose reiktų rinktis algoritmą *BP*, *Branch-Fast* arba *Branch-Const-2*.

Summary

This paper present an experimental study of graph edit distance (GED) heuristic algorithms with the goal of improving the performance and accuracy of state of the art heuristics. First half of this paper defines GED and provides detailed explanation of three heuristic algorithm classes for computing GED. In the second half, experimental study and analysis of its results is given. The main focus of this paper is the algorithm *IPFP* [BDB+18] which is based on local search and produces tightest upper bounds for GED to this date [BBG+20]. In order to improve performance of *IPFP* as well as improve tightness of produced upper bounds, a good initial solution for the local search must be used. Initial GED solution can be relatively quickly computed using heuristic algorithms based on linear sum assignment problem with error correction (LSAPE-GED). This paper shows that the task of choosing an algorithm for finding the initial GED solution isn't trivial. Graph vertex and edge count are not the only factors influencing GED algorithms, graph structure has an effect as well. This paper provides three modified LSAPE-GED algorithms that showed good results in GED computation: *Star-5*, *Branch-Const-2* and *Branch-Const-4*. Experimental study showed, that in case where a fast GED algorithm is needed one should consider *Branch-Const-4*. If tighter upper bounds of GED are needed, local search algorithm *IPFP* should be used for which initial GED solution should be computed using one of LSAPE-GED heuristics. For small and sparse graphs, algorithms *Refine(BP)* or *Refine(Branch-Fast)* should be used, for medium sized graphs algorithm *Star-5* should be used, for large or very dense graphs, algorithms *BP*, *Branch-Fast* and *Branch-Const-2* work the best.

Turinys

Įvadas

Grafai naudojami įvairiose srityse modeliuoti sudėtingas realaus pasaulio duomenų struktūras, pavyzdžiui bioinformatikos ir chemijos srityse, kompiuterinės regos ir vaizdų atpažinimo aplikacijose, socialiniuose tinkluose ir daugelyje kitų aplikacijų, kurių duomenų modelyje svarbios sąveikos ir ryšiai tarp esybių. Nuolat ir sparčiai augančiame grafų duomenų kiekyje atsiranda poreikis palaikyti greitesnę tokių duomenų paiešką duombazėse. Esant dideliam duomenų kiekiui dažnai įsivelia klaidos ir aptinkami netikslumai juose, ar tai būtų dėl žmogaus įvedimo klaidos ar prietaiso, kuris išgavo grafą, matavimo paklaidos. Didelėse duombazėse ar duombazėse su duomenų triukšmu, vykdant paieška leidžiamos tam tikros užklausų paklaidos, kitaip tariant, ieškomi duomenys panašūs į užklausoje apibrėžtus duomenis.

Panašumo matą tarp grafų galima apibrėžti įvairiai, tačiau šiame darbe nagrinėjamas grafų redagavimo atstumas [SF83]. Trumpai tariant, grafų redagavimo atstumas yra reikalingas skaičius grafo pakeitimų transformuoti grafą G į H. Galimi grafo pakeitimai yra grafo viršūnių ar briaunų trynimas, pridėjimas, ar jų žymės keitimas. Kadangi grafų redagavimo atstumą galima taikyti bet kokio tipo grafui ir šis matas leidžia modifikuoti tiek grafo viršūnes tiek jo briaunas, tai vienas populiariausių grafų panašumo matų [ZTW+09].

Nors grafų redagavimo atstumas yra patogi metrika ir gerai atspindi grafų panašumą, jos skaičiavimas priklauso NP-sudėtingų uždavinių klasei [ZTW+09]. Dėl šios priežasties egzistuoja didelis kiekis skirtingų euristinių algoritmų skaičiuoti tikslią grafų redagavimo atstumo vertę ar jos viršutinį bei apatinį rėžius [BBG+20]. Kadangi skirtingų euristinių algoritmų yra daug ir panašių grafų paieškoje vyrauja daug parametrų, yra sunku nustatyti, kuris algoritmas labiausiai tinka konkrečios grafų duombazės užklausoms spartinti.

Esami euristiniai algoritmai grafų redagavimo atstumui aproksimuoti gražina reikšmes, kurios vidutiniškai nutolusios nuo tikslios reikšmės per apytiksliai 2% nuo gautos reikšmės [BBG+20]. Atsižvelgus į tai, kad tikslaus grafų redagavimo atstumo skaičiavimas yra NP-sudėtingas uždavinys, o aproksimacijos apskaičiuojamos per polinominį laiką gražina reikšmes su apytiksle 2% paklaida – tai labai geras ir tuo pačiu tikėtinai sunkiai pagerinamas rezultatas [BBG+20]. Tačiau yra algoritmai, kurie veikia žymiai greičiau nei tiksliausius rezultatus išgaunantys algoritmai, bet gražina tik šiek tiek prastesnius apatinį ir viršutinį grafų redagavimo atstumo rėžius. Vieno iš tokių algoritmų pavyzdys yra algoritmas *Node* [BG17a], kuris veikia keliomis eilėmis greičiau nei tiksliausius apatinius rėžius išgaunantis *ADJ-IP* algoritmas [JH06] ar tiksliausią viršutinį rėžį skaičiuojantis *IPFP* algoritmas [LHS09]. Remiantis tuo, kad yra spraga tarp algoritmų vykdymo laiko ir jų išgaunamų rezultatų tikslumo, galima manyti, kad egzistuoja galimos algoritmų optimizacijos grafų redagavimo atstumo skaičiavimams spartinti.

Šio **magistrinio darbo bendras tikslas**: paspartinti tiksliausius esamus grafo redagavimo atstumo paieškos euristinius algoritmus.

Šiam tikslui pasiekti keliami uždaviniai:

- Išanalizuoti esamus euristinius algoritmus grafų filtravimo ir redagavimo atstumo paieškai.

- Paruošti eksperimentų aplinka esamų grafų paieškos algoritmų efektyvumo matavimui ir palyginimui.
- Atlikti panašių grafų paieškos eksperimentus naudojant grafų paieškos euristikų kombinacijas.
- Išanalizuoti pavienių euristikų ir užduoties kintamųjų priklausomybes grafo redagavimo atstumo tikslumo ir paieškos greičio atžvilgiu.
- Atlikti tiksliausių euristinių algoritmų modifikacijas siekiant paspartinti jų vykdymo laiką.

Šis darbas sudarytas iš penkių dalių. Pirmame skyriuje apibrėžiamas grafų redagavimo atstumas ir susijusios sąvokos, kurios naudojamos tolimesniuose skyriuose. Antrame skyriuje apibrėžiami trys skirtingi metodai grafų redagavimo atstumui aproksimuoti ir identifikuojami svarbiausi šių dienų euristiniai algoritmai. Trečiame skyriuje aprašomi eksperimentuose naudoti grafai ir jų generavimo modeliai. Ketvirtame skyriuje pristatomi šio darbo principiniai sprendimai. Paskutinėje šio darbo dalyje pateikiami ir aprašomi šio darbo eksperimentai bei jų rezultatai.

1. Grafo redagavimo atstumas

Siekiant nustatyti ar du grafai G ir H yra panašūs, reikalingas matas, kuris leistu kiekybiškai įvertinti skirtumą tarp grafų. Šiame darbe nagrinėjamas matas - grafų redagavimo atstumas. Tai nėra vienintelis matas nustatyti grafų panašumui, tačiau dėl savo lankstumo ir universalumo, jis yra vienas populiariausių tarp grafų paieškos ar vaizdų atpažinimo aplikacijų [ZTW+09].

1.1. Bendros grafo savokos

Šiame darbe, neorientuotas žymėtas grafas G yra apibrėžiamas, kaip ketvertas: $G = (V, E, l_V, l_E)$. Čia V yra grafo viršūnių aibė, E yra grafo briaunų aibė, $l_V : V \to \sum^V$ ir $l_E : E \to \sum^E$ yra funkcijos, kurios grafo viršūnėms ir briaunoms priskiria žymas iš atitinkamų abėcėlių. $\sum^V$ ir $\sum^E$ yra grafų viršūnių ir briaunų abėcėlės - žymų aibės. Nežymėta grafą galima traktuoti, kaip grafą, kurio viršūnių ir briaunų žymų abėcėlės sudarytos iš vieno elemento ir funkcijos l_V, l_E visoms viršūnėms bei briaunoms priskiria tą pačią žymą. Toliau, žymų funkcijos l_V ir l_E nėra įtraukiamos į grafų apibrėžimus jeigu jos neturi įtakos duotame kontekste. Visi šiame darbe nagrinėjami grafai turi apibrėžtas, ne tuščias žymų abėcėles $\sum^V$ ir $\sum^E$. Šiame darbe, terminas *grafo dydis* apibrėžiamas grafo viršūnių skaičiumi.

Grafas $G = (V, E)$ yra vadinamas dvidaliu, jei jo viršūnių aibę galima išskaidyti į dvi nepersidengiančias aibes $V = V^1 \cup V^2$, $V^1 \cap V^2 = \emptyset$, , taip kad bet kokia briauna $(u, v) \in E$ jungtų viršūnę iš V^1 su viršūne iš V^2, kitaip sakant visoms briaunoms $(u, v) \in E$ turi būti teisinga $u \in V^1, v \in V^2$. Tokie grafai paprastai žymimi $G_{n,m} = (U, V, E)$, kur U, V yra dvi grafo viršūnių aibės tenkinančios dvidalio grafo apibrėžimą, $n = |U|$ ir $m = |V|$.

Grafai $G = (V_G, E_G)$ ir $H = (V_H, E_H)$ yra vadinami izomorfiškais jeigu egzistuoja tokia apibus vienareikšmė atitiktis tarp šių grafų viršūnių $f : V_G \to V_H$, kad bet kurios dvi viršūnės v_i, v_j priklausančios grafui G bus sujungtos briauna $(v_i, v_j) \in E_G$ tada ir tik tada kai bus sujungtos briauna atitinkamos viršūnės $(f(v_i), f(v_j)) \in E_H$ grafe H.

Grafo $G = (V, E)$ viršūnės $v \in V$ ekscentricitetas apibrėžiamas formule $e(v) = \max_{u \in V} d(v, u)$, kur $d(v, u)$ žymi atstumą tarp viršūnių. Tai yra, viršūnės v ekscentricitetas žymi didžiausia atstumą nuo jos iki likusių grafo viršūnių. Naudojantis grafo viršūnės ekscentriciteto apibrėžimu, galime apibrėžti grafo spindulio bei skersmens sąvokas. Grafo spindulys apibrėžiamas formule $r(G) = \min_{v \in V} e(v)$, tai yra skaičius lygus mažiausiam viršūnių ekscentricitetui. Grafo skersmuo apibrėžiamas formule $d(G) = \max_{v \in V} e(v)$, tai yra skaičius lygus didžiausias viršūnių ekscentricitetui, kuri taip pat galima apibūdinti, kaip ilgiausią, trumpiausią kelią tarp visų galimų viršūnių $v \in V$ ir $u \in V$. Kadangi ne visi šiame darbe nagrinėjami grafai yra jungūs, kitaip sakant grafai sudaryti iš kelių jungių komponenčių, skersmuo bei spindulys tokiems grafams yra skaičiuojami su papildomomis taisyklėmis. Pirma, kiekvienai nejungaus grafo G' komponentei yra surandamas skersmuo bei spindulys. Toliau nejungaus grafo skersmeniui $d(G')$ yra priskiriamas didžiausias skersmuo iš visų grafo komponenčių, ir analogiškai grafo spinduliui $r(G')$ yra priskiriamas didžiausias spindulys iš visų grafo komponenčių.

1.2. Grafo redagavimo seka paremtas apibrėžimas

Kadangi didžioji dalis grafų, kuriems taikomas grafų redagavimo atstumas yra neorientuoti grafai [ARR15; RB08; STF+17], euristiniai algoritmai dažniausiai būna sukurti ir optimizuoti būtent tokio tipo grafams. Tačiau neretai tokius algoritmus galima nesudėtingai modifikuoti, kad jie būtu tinkami skaičiuoti grafo redagavimo atstumą ir orientuotiems grafams.

Grafo redagavimo seka tarp grafų G ir H yra grafo modifikacijų seka $P := (o_1, o_2, \ldots, o_n)$, kurią pritaikius grafui G gaunamas grafas G', kuris yra izomorfinis grafui H. Čia o_i yra viena iš šešių galimų grafo modifikacijų: neprijungtos viršūnės pridėjimas, neprijungtos viršūnės ištrynimas, viršūnės žymos keitimas, briaunos pridėjimas, briaunos išmetimas ir briaunos žymos keitimas.

Duotai grafo redagavimo sekai P galima priskirti redagavimo kainą $c(P)$ - skaitinę vertę simbolizuojančia grafo transformacijos kainą. Grafo redagavimo sekos P kaina apibrėžiama susumavus visas pavienių grafo modifikacijų kainas: $c(P) := \sum_{i=1}^{n} c(o_i)$. Svarbu pastebėti, kad skirtingi grafo modifikacijų tipai ar konkrečios operacijos gali turėti skirtingas susietas redagavimo kainas - pavyzdžiui viršūnės pridėjimas gali turėti didesnę skaitinę vertę, nei briaunos tarp egzistuojančių viršūnių pridėjimas. Jei visos grafo redagavimo operacijų kainos lygios vienetui: $c(o_i) = 1, i \in (1..n)$, tai tokia redagavimų kainų priskyrimo funkcija vadinama pastovia. Grafo modifikacijų kainos nėra apribotos konstantomis, jos gali būti modeliuojamos įvairiai, pavyzdžiui rašto simbolių atpažinimo uždaviniuose briaunos pridėjimo ar ištrynimo operacija galima būtų modeliuoti, kaip atstumą tarp viršūnių euklidinėje erdvėje. Grafo modifikacijų kainos modelis priklauso nuo srities ar aplikacijos, kurioje taikomi grafų redagavimo atstumo skaičiavimai.

Grafų redagavimo atstumą tarp grafų G ir H galima apibrėžti taip:

$$GRA(G, H) := \min\{\, c(P) \mid P \in \Psi(G, H) \,\} \tag{1}$$

Čia $c(P)$ yra grafo redagavimo sekos kaina, o $\Psi(G, H)$ yra aibė visų galimų grafo redagavimo sekų tarp grafų G ir H. Šis apibrėžimas yra paprastas ir intuityvus, tačiau neduoda daug naudos algoritmų kontekste ir jų konstravime. Pirma, šis apibrėžimas yra paremtas grafų izomorfizmu, o nustatyti ar du grafai yra izomorfiški per polinomai laika neįmanoma [Bab16], kitaip sakant grafų izomorfizmo uždavinys priklauso NP uždavinių aibei. Taip pat per polinominį laiką neįmanoma patikrinti ar grafo redagavimo seka P transformuoja grafą G į grafą H, nekalbant apie tai, kad šis grafo redagavimo atstumo apibrėžimas sufleruoja perrinkti visas galimas grafo redagavimo sekas. Dėl šių priežasčių dažniau yra naudojamas alternatyvus grafo redagavimo atstumo apibrėžimas, kuris yra paremtas grafo viršūnių atvaizdžiu.

1.3. Grafų viršūnių atvaizdžiu paremtas apibrėžimas

Grafų viršūnių atvaizdis tarp grafų G ir H yra apibrėžiamas, kaip sąryšis (angl. relation) $\pi \subseteq V_{G+} \times V_{H+}$ jei ir tik jei visiems $u \in V_G$ teisinga $|\{v \mid v \in V_{H+} \wedge (u, v) \in \pi\}| = 1$ ir visiems $v \in V_H$ teisinga $|\{u \mid u \in V_{G+} \wedge (u, v) \in \pi\}| = 1$, kitaip sakant, viršūnių atvaizdis π turi padengti visas tikras viršūnes iš grafų G ir H lygiai vieną kartą. Čia $V_{G+} := V_G \cup \{\epsilon\}$ ir $V_{H+} := V_H \cup \{\epsilon\}$,

kur ϵ yra papildoma, fiktyvi grafo viršūnė. $\Pi(G, H)$ yra aibė visų galimų viršūnių atvaizdžių tarp grafų G ir H.

Toliau indeksai i, j bus naudojami kreiptis į viršūnes iš grafo G, o indeksai k, l kreiptis į grafo H viršūnes. Trumpinys $\pi(i) = k$ bus naudojamas jei ir tik jei $(i, k) \in \pi$ ir $i \neq \epsilon$. Trumpinys $\pi^{-1}(k) = i$ bus naudojamas jei ir tik jei $(i, k) \in \pi$ ir $k \neq \epsilon$. Grafų briaunoms $e = (i, j) \in E^G$ ir $f = (k, l) \in E^H$ bus taikomi trumpiniai $\pi(e) := (\pi(i), \pi(j))$ ir $\pi^{-1}(f) := (\pi^{-1}(k), \pi^{-1}(l))$.

Neakivaizdu, tačiau iš viršūnių atvaizdžio $\pi \in \Pi(G, H)$ galima atsekti kokios modifikavimo operacijos buvo atliktos kiekvienai viršūnei ir briaunai iš grafo G. Jei atvaizdis π atvaizduoja tikrą viršūnę i į fiktyvia viršūnę ϵ - viršūnė i buvo ištrinta. Analogiškai, jei π atvaizduoja fiktyvia viršūnę ϵ į tikrą viršūnę k - viršūnė k buvo pridėta. Jei atvaizduojama tikra viršūnė i į kitą tikrą viršūnę k - viršūnės žymė buvo pakeista iš $l_V^G(i)$ į $l_V^H(k)$. Jeigu $\pi(i) = k$, $\pi(j) = l$ ir $(i, j) \in E^G$, bet $(k, l) \notin E^H$, tai briauna (i, j) buvo ištrinta. Analogiškai jei $(i, j) \notin E^G$, bet $(k, l) \in E^H$, tai briauna (k, l) buvo pridėta. Galiausiai jei abi briaunos egzistuoja grafuose: $(i, j) \in E^G$, $(k, l) \in E^H$, tai briaunos žymė buvo pakeista iš $l_E^G((i, j))$ į $l_E^H((k, l))$.

Apibrėžiamos grafo viršūnių bei briaunų redagavimo kainų funkcijos, kurios kiekvienai grafo modifikavimo operacijai priskiria realią, neneigiamą skaitinę vertę: $c_V := (\sum_V \cup \{\epsilon\})^2 \to \mathbb{R}_{\geq 0}$ ir $c_E := (\sum_E \cup \{\epsilon\})^2 \to \mathbb{R}_{\geq 0}$. Čia $\sum_V$ ir $\sum_E$ yra bendros grafų viršūnių bei briaunų žymų abėcėlės, o ϵ yra papildoma, fiktyvi grafo viršūnė ar briauna. Visos šešios galimos grafo operacijos ir susijusios kainų funkcijos detaliau pavaizduotos lentelėje 1.

Atvaizdžio atvejis	**Operacija**	**Operacijos kaina**
Viršūnių operacijos		
$\pi(i) = k$	Pakeisti viršūnės žymą	$c_V(i, k) := c_V(l_V^G(i), l_V^H(k))$
$\pi(i) = \epsilon$	Ištrinti viršūnę i	$c_V(i, \epsilon) := c_V(l_V^G(i), \epsilon)$
$\pi^{-1}(k) = \epsilon$	Pridėti viršūnę k	$c_V(\epsilon, k) := c_V(\epsilon, l_V^H(k))$
Briaunų operacijos		
$\pi(e) = f$	Pakeisti briaunos žymą	$c_E(e, f) := c_E(l_E^G(e), l_E^H(f))$
$\pi(e) \notin E^H$	Ištrinti briauną e	$c_E(e, \epsilon) := c_E(l_E^G(e), \epsilon)$
$\pi^{-1}(f) \notin E^G$	Pridėti briauną f	$c_E(\epsilon, f) := c_E(\epsilon, l_E^G(f))$

1 lentelė. Grafų viršūnių atvaizdžio žymėjimai ir susietos redagavimo kainos

Turint grafo viršūnių atvaizdžio apibrėžimą, galima apibrėžti indukuotą redagavimo seką, kuri išplaukia iš duoto atvaizdžio. Tegul duotiems grafams G ir H sukonstruotas viršūnių atvaizdis $\pi \in \Pi(G, H)$ ir O yra aibė grafo operacijų, kurias galima dekoduoti iš atvaizdžio π remiantis lentele 1. Tada aibės O išrikiavimas $P_\pi := (o_i)_{i=1}^{|O|}$ yra apibrėžiamas, kaip grafo viršūnių atvaizdžio π indukuota redagavimo seka, jei ir tik jei briaunų trynimo operacijos yra pirmos išrikiuotoje sekoje ir briaunų pridėjimo operacijos yra paskutinės. Esant tokiems apribojimams galima parodyti, kad indukuotos redagavimo sekos yra tikros redagavimo sekos, kitaip sakant $P_\pi \in \Psi(G, H)$ galioja visiems $\pi \in \Pi(G, H)$ [BBC+17].

Pasinaudojus grafo viršūnių atvaizdžiais ir jų indukuotomis redagavimo sekomis, galima apibrėžti grafo redagavimo atstumą taip:

$$GRA(G, H) := \min\{\, c(P_\pi) \mid \pi \in \Pi(G, H) \,\} \tag{2}$$

Čia $P_\pi \in \Pi(G, H)$ yra indukuota redagavimo seka, kuri buvo išgauta iš grafo G viršūnių atvaizdžio π. Pateikti grafų redagavimo atstumo apibrėžimai (1) ir (2) yra ekvivalentūs jeigu juose naudojamų grafo redagavimo operacijų kainų funkcijos c_V ir c_E tenkina atstumo funkcijos reikalavimus [JH06]. Pagrindinis privalumas apibrėžimo (2) lyginant su apibrėžimu (1) - grafo viršūnių atvaizdžius galima trivialiai konstruoti, lyginant su netrivialiu grafo redagavimo kelių konstravimu. Taip pat iš kiekvieno viršūnių atvaizdžio $\pi \in \Pi(G, H)$ galima apskaičiuoti viršutinį grafų redagavimo atstumo rėžį susumavus indukuotos redagavimo sekos kainą: $c(P_\pi) \geq GRA(G, H)$ [BBG+20].

2. Grafų redagavimo atstumo aproksimacija

Apskaičiuoti tikslią grafų redagavimo atstumo reikšmę tarp duotų grafų yra sudėtinga problema, kuri priklauso NP-sudėtingų uždavinių klasei. Grafų redagavimo atstumo skaičiavimas nepalengvėja savo kompleksiškumu, net jeigu naudojamos grafo redagavimo funkcijos yra pastovios - visos grafo viršūnių ar briaunų redagavimo operacijos įvertinamos ta pačia skaitine reikšme, pavyzdžiui vienetu [ZTW+09].

Dėl tikslaus grafų redagavimo atstumo skaičiavimo sudėtingumo, egzistuoja daug skirtingų euristinių algoritmų, kurie aproksimuoja grafo redagavimo atstumą per viršutinį ar apatinį rėžį. Euristikos naudoja įvairius metodus atlikti skaičiavimams, pavyzdžiui tiesinės sumos priskyrimo su klaidų taisymu (angl. linear sum assignment problem with error correction), tiesinio programavimo ar lokalios paieškos metodus. Būtent šie trys metodai vyrauja didžiojoje dalyje euristinių algoritmų grafų redagavimo atstumo aproksimacijai [BBG+20], todėl toliau nagrinėjami euristiniai algoritmai yra priskiriami vienai iš šių klasių:

- **TSPKT-GRA** - Euristikos, kurios naudoja tiesinės sumos priskyrimo su klaidų taisymo metodus.
- **TP-GRA** - Euristikos, kurios naudoja tiesinio programavimo metodus.
- **LP-GRA** - Euristikos, kurios naudoja lokalios paieškos metodus.

Euristiniai algoritmai, kurie nenaudoja aukščiau išvardintų metodų grafų redagavimo atstumo skaičiavimams yra priskiriami maišytų algoritmų klasei.

2.1. Euristinių algoritmų kategorija TSPKT-GRA

Grafų redagavimo atstumo apibrėžimas, kuris remiasi viršūnių atvaizdžiais, yra nesudėtingai transformuojamas ir pritaikomas tiesinės sumos priskyrimo uždaviniui spręsti. Neformaliai apibrėžiant, tiesinės sumos priskyrimo uždavinys reikalauja duotai $n \times m$ dydžio kainų matricai $C = (c_{ij})$ priskirti kiekvieną matricos eilutę vienam matricos stulpeliui taip, kad atitinkamos matricos elementų skaitinių reikšmių suma būtų kaip įmanoma mažesnė.

Algoritmai, kurie sprendžia tiesinės sumos priskyrimo uždavinius ar jų modifikacijas yra glaudžiai susiję su grafo redagavimo atstumo paieška. Grafo viršūnių atvaizdį $\pi \in \Pi(G, H)$ galima trivialiai transformuoti į ir iš kainų matricos C. Radus optimalų ar godų kainų matricos C sprendinį tiesinės sumos priskyrimo užduočiai, sprendinį galima transformuoti į viršūnių atvaizdį π ko pasėkoje gaunamas viršutinis grafų redagavimo atstumo rėžis.

2.1.1. Tiesinės sumos priskyrimo uždavinys

Tiesinės sumos priskyrimo uždavinį galima apibrėžti pasinaudojus dvidalio grafo apibrėžimu. Tarkim duotas dvidalis grafas $G_{n,m} = (U, V, U \times V)$, kur $U = \{u_1, u_2, \ldots, u_n\}$ ir $V = \{v_1, v_2, \ldots, v_m\}$ yra dvi nepersidengiančios grafo G viršūnių aibės, ir $n < m$. Briaunų aibė $M \subset U \times V$ vadinama pilnu deriniu jei ir tik jei visos viršūnės iš U sujungtos tik su viena

briauna iš M ir visos viršūnės iš V yra sujungtos tik su viena briauna iš M. Pilną briaunų derinį M galima užrašyti binarine matrica $\overline{X} = (\overline{x}_{i,j}) \in \{0,1\}^{n \times m}$, kur $\overline{x}_{i,j} = 1$ reiškia $(u_i, v_j) \in M$. Aibė visų galimų pilnų derinių grafui $G_{n,m}$ yra $\Pi_{n,m}$.

Duotai kainų matricai $\overline{C} = (\overline{c}_{i,j}) \in \mathbb{R}^{n \times m}$, tiesinės sumos priskyrimo uždavinys reikalauja rasti pilną derinį $\overline{X}^* \in \Pi_{n,m}$ tokį, kad būtų minimizuota kainos matrica C:

$$TSP(\overline{C}) = \overline{X}^* \in argmin_{\overline{X} \in \Pi_{n,m}} L(\overline{X}, \overline{C}) \tag{3}$$

Čia $L(\overline{X}, \overline{C}) = \sum_{i=1}^{n} \sum_{j=1}^{m} \overline{c}_{i,j} \overline{x}_{i,j}$ - kainų matricos $\overline{C}$ elementų suma, sumuojant pagal binarinę matricą $\overline{X}$. Šį uždavinį galima išspręsti per polinominį laiką pasinaudojus keletą egzistuojančių algoritmų. Pavyzdžiui jeigu kainų matrica $\overline{C}$ yra subalansuota, kitaip sakant $n = m$, tai tiesinės sumos priskyrimo uždavinio, naudojantis Munkreso Vengrų algoritmų [Kuh56; Mun57], sudėtingumas laiko atžvilgiu lygus $O(n^3)$ ir $O(n^2)$ atminties atžvilgiu. Bendru atveju, šio uždavinio laiko sudėtingumas lygus $O(n^2 m)$ [BL71].

2.1.2. Tiesinės sumos priskyrimo su klaidų taisymu uždavinys

Duotam pilnam briaunų deriniui $\overline{X}$ iš dvidalio grafo $G_{n,m} = (U, V, U \times V)$ ir kainų matricos $\overline{C} \in \mathbb{R}^{n \times m}$, binarinės matricos $\overline{X}$ elementas $\overline{x}_{i,j} = 1$ traktuojamas, kaip grafo viršūnės $u_i \in U$ keitimas į viršūnę $v_j \in V$ su kaina $\overline{c}_{i,j}$. Sekant tai, tiesinės sumos priskyrimo uždavinį galima traktuoti, kaip viršūnių iš aibės U priskyrimą viršūnėms iš aibės V su minimalia galima kaina. Reiktų atkreipti dėmesį, kad tiesinės sumos priskyrimo uždavinys nereikalauja, kad visos viršūnės iš aibės V būtų susietos su viršūne iš aibės U, kadangi $n < m$ ir viršūnė iš U gali būti susieta tik su viena viršūne iš V. Dėl šio apribojimo egzistuoja praplėstas šio uždavinio variantas - tiesinės sumos priskyrimo su klaidų taisymu uždavinys (toliau TSPKT).

Skaičiuojant panašumo matą, kaip grafų redagavimo atstumą, tarp skirtingų grafų G ir H neretai tenka ne tik pervadinti atitinkamas grafo viršūnes iš duotų grafų, bet ir ištrinti ar pridėti naują viršūnę. TSPKT uždavinys aprėpia šiuos reikalavimus, kitaip sakant visos viršūnės iš aibės V yra padengiamos nepriklausomai nuo viršūnių aibės U dydžio. Viršūnių aibės U ir V yra praplečiamos su viena papildoma, fiktyvia viršūne: $U^+ := U \cup \{\epsilon\}$ ir $V^+ := V \cup \{\epsilon\}$. Briaunų aibė $M = S \cup R \cup I \subset U^+ \times V^+$ yra vadinama klaidas taisantis derinys daliniam grafui $G_{n,m,\epsilon} = (U^+, V^+, U^+ \times V^+)$ jei ir tik jei $(\epsilon, \epsilon) \notin M$ ir visos viršūnės iš aibių U ir V yra sujungtos tik su viena briauna iš M. Čia $S \subset U \times V$ simbolizuoja viršūnes iš U, kurios keičiamos į viršūnes iš V, $R \subset U \times \{\epsilon\}$ simbolizuoja viršūnes, kurios ištrinamos iš U ir $I \subset \{\epsilon\} \times V$ simbolizuoja viršūnes, kurios pridedamos į V. Aibė visų galimų klaidas taisančių derinių grafui $G_{n,m,\epsilon}$ yra $\Pi_{n,m,\epsilon}$, kuri apsibrėžia taip:

$$\Pi_{n,m,\epsilon} = \Bigg\{ X = \begin{pmatrix} X_S & X_R \\ X_I & 0 \end{pmatrix} \in \{0,1\}^{(n+1)\times(m+1)} : \\ \forall j \in (1..m),\ x_{\epsilon,j} + \sum_{i=1}^{n} x_{i,j} = 1, \\ \forall i \in (1..n),\ x_{i,\epsilon} + \sum_{j=1}^{m} x_{i,j} = 1 \Bigg\}$$

Kur $X_S = (x_{i,j}) \in \mathbb{R}^{n\times m}$ žymi grafo viršūnių keitimo operacijas, $X_R = (x_{i,\epsilon}) \in \mathbb{R}^{n\times 1}$ žymi grafo viršūnių trynimo operacijas ir $X_I = (x_{\epsilon,j}) \in \mathbb{R}^{1\times n}$ žymi grafo viršūnių pridėjimo operacijas. Kitaip sakant, klaidas taisantis briaunų derinys X yra pilnas briaunų derinys $\overline{X}$ su papildoma eilute ir stulpeliu simbolizuojančius papildoma viršūnę ϵ. Analogiškai apibrėžiama kainų matrica $C \in \mathbb{R}^{(n+1)\times(m+1)}$:

$$C = \begin{pmatrix} C_S & C_R \\ C_I & 0 \end{pmatrix} \tag{4}$$

Kur $C_S = (c_{i,j}) \in \mathbb{R}^{n\times m}$ žymi grafo viršūnių keitimo kainas, $C_R = (c_{i,\epsilon}) \in \mathbb{R}^{n\times 1}$ žymi grafo viršūnių trynimo kainas ir $C_I = (c_{\epsilon,j}) \in \mathbb{R}^{1\times n}$ žymi grafo viršūnių pridėjimo kainas.

Turint klaidas taisančio derinio ir kainų matricos apibrėžimus, galima apibrėžti tiesinių sumų priskyrimo su klaidų taisymu uždavinio formuluote. Duotai kainų matricai $C \in \mathbb{R}^{(n+1)\times(m+1)}$ TSPKT uždavinys reikalauja rasti klaidas taisantį derinį $X^* \in \Pi_{n,m,\epsilon}$, tokį, kad būtų minimizuota kainos matrica C.

$$TSPKT(C) = X^* \in argmin_{X\in\Pi_{n,m,\epsilon}} L(X,C) \tag{5}$$

Čia $L(X,C) = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{i,j}x_{i,j} + \sum_{i=1}^{n} c_{i,\epsilon}x_{i,\epsilon} + \sum_{j=1}^{m} c_{\epsilon,j}x_{\epsilon,j}$ - kainų matricos C elementų suma, sumuojant pagal binarinę matricą X.

2.1.3. Algoritmų šablonas

Grafo viršūnių atvaizdžiai $\pi \in \Pi(G,H)$ yra glaudžiai susiję su tiesinės sumos priskyrimo su klaidų taisymo uždavinio sprendiniais (toliau TSPKT). Imkime grafus G, H ir TSPKT užduotį $C \in \mathbb{R}^{(|V_G|+1)\times(|V_H|+1)}$. Atsižvelgus i grafo viršūnių atvaizdžio ir TSPKT apibrėžimus, galima įžvelgti, kad grafo viršūnių atvaizdžių aibė $\Pi(G,H)$ grafams G ir H dera su TSPKT galimų sprendinių aibe $\Pi_{|V_G|,|V_H|,\epsilon}$ užduočiai C. Visiems $i \in (1..|V_G|)$ ir $k \in (1..|V_H|)$ iš C galima susieti i-tają eilutę su grafo viršūnę $u_i \in V_G$ ir j-tajį stulpelį su grafo viršūnę $v_j \in V_H$. Paskutinė eilutė ir stulpelis iš C susiejamas su fiktyvia viršūne ϵ. Tokiu būdu, visus galimus TSPKT sprendinius X užduočiai C galima interpretuoti, kaip grafo viršūnių atvaizdžius ir gauti viršutinį grafų redagavimo atstumo rėžį apskaičiavus indukuotos pakeitimų sekos $c(P_X)$ kainą.

Euristiniai algoritmai, kurie remiasi TSPKT formuluote grafų redagavimo atstumui skaičiuoti, apibendrinami abstrakčiu algoritmų šablonu, kuris pateiktas iliustracijoje Algoritmas 1. Pirmame

žingsnyje euristiniai algoritmai iš duotų grafų G, H ir redagavimo kainų funkcijų c_V, c_E sukonstruoja TSPKT uždavinį $C \in \mathbb{R}^{(|V_G|+1)\times(|V_H|+1)}$, tokį kad optimalus jo sprendinys indukuotų kuo pigesnę redagavimo seką tarp grafų G ir H. Uždavinio C konstravimas yra esminė dalis, kur skirtingi euristiniai algoritmai ir jų našumas laiko atžvilgiu išsiskiria. Toliau TSPKT uždavinys C yra išsprendžiamas optimaliu arba godžiu algoritmu ir redagavimo kainą $c(P_X)$, gauta iš indukuotos redagavimo sekos radus TSPKT sprendinį X, yra priskiriama, kaip viršutinis grafų redagavimo atstumo rėžis. Jeigu protokolas pagal kurį buvo konstruojamas TSPKT uždavinys C garantuoja, kad esant optimaliam C sprendiniui X^*, galima apibrėžti mastelio keitimo funkciją $\xi(G, H, c_V, c_E)$ tokią kad nelygybė $\xi(G, H, c_V, c_E) \cdot c(P_{X^*}) \leq GRA(G, H)$ yra patenkinta visiems grafams ir redagavimo funkcijoms bei optimalus TSPKT algoritmas buvo naudotas apskaičiuoti sprendiniui X, tai $\xi(G, H, c_V, c_E) \cdot c(P_X)$ yra gražinamas, kaip apatinis grafų redagavimo atstumo rėžis. Kitu atveju tik viršutinis rėžis yra gražinamas.

Algoritmas 1 TSPKT-GRA euristikų šablonas

procedure TSPKT-GRA(G, H, c_V, c_E)
    Remiantis įvestimi sukonstruojamas TSPKT uždavinys $C \in \mathbb{R}^{(|V_G|+1)\times(|V_H|+1)}$
    Naudojantis optimaliu ar godžiu TSPKT algoritmu randamas C sprendinys X
    $VR \leftarrow c(P_X)$ ▷ Apskaičiuojamas viršutinis GRA rėžis
    if eilutė 2 užtikrina $\xi(G, H, c_V, c_E) \cdot c(P_{X^*}) \leq GRA(G, H)$ **then**
        if optimalus TSPKT algoritmas buvo naudotas eilutėje 2 **then**
            $AR \leftarrow \xi(G, H, c_V, c_E) \cdot c(P_X)$ ▷ Apskaičiuojamas apatinis GRA rėžis
            return AR ir VR
        else
            return VR
        end if
    end if
    return VR
end procedure

2.2. Euristinių algoritmų kategorija TP-GRA

Grafų redagavimo atstumą galima apibrėžti, kaip tiesinio programavimo uždavinį. Pavyzdžiui apibrėžimą 2, kuris apibrėžtas, kaip minimizavimo uždavinys aibei visų galimų viršūnių atvaizdžių tarp grafų G ir H, galima transformuoti į kvadratinio programavimo formuluotę grafų redagavimo atstumui rasti. Iš tiesų grafų redagavimo atstumo uždavinys priklauso kvadratinio programavimo uždavinių šeimai [Rie15], kuri pati priklauso NP-pilnų uždavinių aibei. Euristiniai algoritmai paprastai ištiesina kvadratinio programavimo uždavinį ir sprendžia tiesinio programavimo uždavinį siekiant rasti apatinį grafų redagavimo atstumo rėžį.

2.2.1. Kvadratinio programavimo uždavinys

Kvadratinio programavimo uždaviniai sprendžia bėda, kaip priskirti n elementų iš pirmos aibės $S = \{s_1, s_2, \ldots, s_n\}$, elementams iš antros aibės $Q = \{q_1, q_2, \ldots, q_n\}$ laikantis tam tikrų priskyrimo apribojimų, kurie dažnai reikalauja sudėtingų skaičiavimų. Formaliai apibrėžti aibės

S elementų priskyrimą aibei Q galima perfrazuojant priskyrimo uždavinį, kaip sveikųjų skaičių aibės $(1, 2, \ldots, n)$ išrikiavimą į aibę $(\phi_1, \phi_2, \ldots, \phi_n)$. Taip išrikiuotos aibės $(\phi_1, \phi_2, \ldots, \phi_n)$ elementus galima traktuoti, kaip elemento $s_1 \in S$ priskyrimą elementui $q_{\phi_1} \in Q$, antro elemento $s_2 \in S$ priskyrimą elementui $q_{\phi_2} \in Q$ ir taip toliau.

Norint performuluoti grafų redagavimo atstumo apibrėžima remiantis kvadratinio programavimo uždaviniu reikia pirma išspręsti dvi bėdas. Pirma, kvadratinio programavimo uždaviniai taikomi vienodo dydžio aibėms $|S| = |Q|$, tuo tarpu grafai, kuriems bus skaičiuojamas redagavimo atstumas, gali tūrėti skirtingą kiekį viršūnių ar briaunų. Antra, kvadratinio programavimo uždaviniai kiekvieną aibės S elementą priskiria vienam elementui iš aibės Q - kitaip sakant yra sudaroma bijekcija tarp duotų aibių. Grafų redagavimo atstumas yra bendresnis priskyrimo uždavinys, kur galimos viršūnių trynimo ar pridėjimo operacijos, o ne tik viršūnių keitimo, žymės pervadinimo, operacijos. Tačiau šiuos du nederančius reikalavimus galima nesudėtingai apeiti praplečiant grafų G, H viršūnių aibes su papildomomis, fiktyviomis viršūnėmis: $V_{G+} = V_G \cup \{\epsilon_1, \epsilon_2, \ldots, \epsilon_m\}$ ir $V_{H+} = V_H \cup \{\epsilon_1, \epsilon_2, \ldots, \epsilon_n\}$, kur $n = |V_G|$ ir $m = |V_H|$. Kadangi abu grafai G ir H dabar turi po vienodą skaičių viršūnių: $|V_{G+}| = |V_{H+}| = n + m$, grafų viršūnių gretimumo matricos A ir B taip pat bus vienodo dydžio. Remiantis išplėstomis viršūnių aibėmis V_{G+} ir V_{H+} galime apibrėžti redagavimo kainų matricą C:

$$C = \begin{array}{c|cccc|cccc} & v_1 & v_2 & \ldots & v_m & \epsilon_1 & \epsilon_2 & \ldots & \epsilon_n \\ \hline u_1 & c_{11} & c_{12} & \ldots & c_{1m} & c_{1\epsilon} & \infty & \ldots & \infty \\ u_2 & c_{21} & c_{22} & \ldots & c_{2m} & \infty & c_{2\epsilon} & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \infty \\ u_n & c_{n1} & c_{n2} & \ldots & c_{nm} & \infty & \ldots & \infty & c_{n\epsilon} \\ \hline \epsilon_1 & c_{\epsilon 1} & \infty & \ldots & \infty & 0 & 0 & \ldots & 0 \\ \epsilon_1 & \infty & c_{\epsilon 2} & \ddots & \vdots & 0 & 0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \infty & \vdots & \ddots & \ddots & 0 \\ \epsilon_m & \infty & \ldots & \infty & c_{\epsilon m} & 0 & \ldots & 0 & 0 \end{array}$$

Čia matricos C elementas c_{ij} žymį viršūnių keitimo $(u_i \to v_j)$ kainą $c(u_i \to v_j)$, $c_{i\epsilon}$ žymi viršūnės trynimo $(u_i \to \epsilon)$ kainą $c(u_i \to \epsilon)$ ir $c_{\epsilon j}$ žymi naujos viršūnės pridėjimo $(\epsilon \to v_j)$ kainą $c(\epsilon \to v_j)$. Sekant tai, viršutinė kairė matricos C dalis žymi visų galimų viršūnių keitimų kainas, viršutinės dešiniosios dalies įstrižainė žymi visas galimas viršūnių trynimo kainas ir galiausiai apatinės kairiosios dalies įstrižainė žymi visas galimas naujų viršūnių pridėjimo kainas. Kadangi bet kokią viršūnę galima ištrinti ar pridėti tik vieną kartą, viršutinės dešiniosios ir apatinės kairiosios matricos dalies elementams, išskyrus įstrižainės elementus, yra priskirta begaline redagavimo kaina. Taip pat fiktyvios viršūnės keitimas į kitą fiktyvia neturėtų turėti jokios įtakos redagavimo kainai $c(\epsilon \to \epsilon) = 0$, todėl apatinė dešinioji matricos dalis yra užpildyta nuliais.

Duotiems grafams G, H su praplėstomis viršūnių aibėmis V_{G+}, V_{H+} ir atitinkamomis grafų viršūnių gretimumo matricomis A, B, galime apibrėžti kvadratinio programavimo užduoti grafų redagavimo atstumui rasti taip:

$$(\phi_1, \ldots, \phi_{n+m}) = argmin_{(\phi_1,\ldots,\phi_{n+m})\in\rho_{n+m}} \left[\sum_{i=1}^{n+m} c_{i\phi_i} + \sum_{i=1}^{n+m}\sum_{j=1}^{n+m} c(a_{ij} \to b_{\phi_i\phi_j}) \right] \quad (6)$$

Čia $\rho_{(n+m)}$ yra visi galimi sveikųjų skaičių aibės $(1, 2, \ldots, (n+m))$ išrikiavimai - išviso $(n+m)!$ galimų variantų. Pirmasis optimizavimo uždavinio narys $\sum_{i=1}^{n+m} c_{i\phi_i}$ simbolizuoja visas grafo viršūnių operacijas, o antrasis narys $\sum_{i=1}^{n+m}\sum_{j=1}^{n+m} c(a_{ij} \to b_{\phi_i\phi_j})$ simbolizuoja visas grafo briaunų operacijos, kurias galima nesudėtingai atsekti iš perstatymų aibės $(\phi_1, \ldots, \phi_{n+m})$ - jei viršūnė $u_i \in V_{G+}$ priskiriama viršūnei $v_{\phi_i} \in V_{H+}$ ir viršūnė $u_j \in V_{G+}$ priskiriama viršūnei $v_{\phi_j} \in V_{H+}$, tai briauna $(u_i, u_j) \in E^G \cup \{\epsilon\}$ turi būti priskiriama briaunai $(v_{\phi_i}, v_{\phi_j}) \in E^H \cup \{\epsilon\}$.

Optimalus išrikiavimas $(\phi_1, \ldots, \phi_{(n+m)})$ atitinka viršūnių priskyrimo bijekcija $\pi = \{(u_1 \to v_{\phi_1}), (u_2 \to v_{\phi_2}), \ldots, (u_{n+m} \to v_{\phi_{n+m}})\}$, kur kiekvienas viršūnės priskyrimas yra vienas iš: $(u_i \to v_j)$, $(u_i \to \epsilon)$, $(\epsilon \to v_j)$, $(\epsilon \to \epsilon)$. Paskutinį priskyrimo varianta galima atfiltruoti, nes jis nemodifikuoja tikrų grafo viršūnių. Čia galima pastebėti, kad išrikiavimas $(\phi_1, \ldots, \phi_{(n+m)})$ pilnai atitinka indukuotą grafo redagavimo seką P_π iš viršūnių atvaizdžio $\pi \in \Pi(G, H)$ [Rie15]. Taip pat galima daryti išvada, kad suradus optimalų sprendinį duotai kvadratinio programavimo uždavinio formuluotei, taip pat randamas ir grafų redagavimo atstumas.

2.2.2. Algoritmų šablonas

Euristinius algoritmus, kurie remiasi kvadratinio programavimo formuluote grafų redagavimo atstumui skaičiuoti, galima apibendrinti vienu abstrakčiu algoritmų šablonu, kuris pateiktas iliustracijoje Algoritmas 2. Pirmame žingsnyje kvadratinio programavimo grafų redagavimo atstumui rasti formuluotė F yra ištiesinama ir gaunama įvairių sveikų skaičių tiesinio programavimo (angl. mixed integer linear programming) formuluotė $\widehat{F}$ [BGG+71]. Šis F ištiesinimo žingsnis yra esminė grafų redagavimo atstumo aproksimacijos dalis, dėl kurios skirtingi euristiniai algoritmai išsiskiria vienas nuo kito. Toliau visi integralumo apribojimai formuluotėje $\widehat{F}$ yra atpalaiduojami tol kol gaunamas tiesinio programavimo uždavinys F [BGG+71]. Galiausiai F yra išsprendžiamas ir gautas optimalus F sprendinys yra priskiriamas apatiniui grafų redagavimo atstumo rėžiui.

Paprastai tiesinio programavimo formuluotės grafo redagavimo atstumui rasti gražina tik apatinį rėžį, tačiau iš gauto optimalaus tiesinio programavimo uždavinio F galima sukonstruoti TSPKT uždavinį $C \in \mathbb{R}^{(|V_G|+1)\times(|V_H|+1)}$ ir jam radus optimalų sprendinį X^*, indukuotas redagavimo sekos kaina $c(P_{X^*})$ yra priskiriama viršutiniam grafų redagavimo atstumo rėžiui.

Teoriškai, tiesinio programavimo uždavinį F galima išspręsti per $O(var(F)^{3.5} + enc(F))$ laiko sudėtingumą, kur $var(F)$ yra kintamųjų skaičius ir $enc(F)$ yra bitų skaičius reikalingas užkoduoti F [Kar84]. Praktikoje yra paprastai naudojami asimptotiškai lėtesni algoritmai, tačiau praktiškai apskaičiuojantys F greičiau nei tradiciniai, minėto laiko sudėtingumo, algoritmai [BBG+20].

Algoritmas 2 TP-GRA euristikų šablonas

procedure TP-GRA(G, H, c_V, c_E)
    Remiantis įvestimi ištiesinama kvadratinio programavimo formuluotė $\widehat{F}$
    $\widehat{F}$ atpalaiduojama tol kol gaunamas tiesinio programavimo uždavinys F
    $AR \leftarrow solve(F)$ ▷ Išsprendžiamas F ir gaunamas apatinis GRA rėžis
    Remiantis F sprendiniu sukonstruojamas TSPKT uždavinys $C \in \mathbb{R}^{(|V_G|+1)\times(|V_H|+1)}$
    $X^* \leftarrow argmin_{X \in \Pi_{n,m,\epsilon}} L(X, C)$ ▷ Randamas optimalus C sprendinys
    $VR \leftarrow c(P_{X^*})$ ▷ Priskiriamas viršutinis GRA rėžis
    return AR ir VR
end procedure

2.3. Euristinių algoritmų kategorija LP-GRA

Grafų redagavimo atstumą tarp grafų G ir H galima apskaičiuoti pasitelkus lokalios paieškos algoritmu. Egzistuoja ne vienas euristinis algoritmas, kurio pagrindinė komponentė yra tam tikras lokalios paieškos algoritmas. Visos euristikos, kurios pagrįstos lokalia paieška, gražina tik viršutinį grafų redagavimo atstumo rėžį, tačiau jas galima taikyti su neapriboto pobūdžio grafo redagavimo kainų funkcijomis c_V, c_E. Tokių euristinių algoritmų vykdymo laikas ir aproksimacijos tikslumas stipriai priklauso nuo pradinės inicializacijos prieš pradedant paiešką.

2.3.1. Lokalios paieškos algoritmas

Lokali paieška yra optimizavimo algoritmas, kuris paremtas iteratyviu einamojo sprendinio transformavimu siekiant rasti geresnį sprendinį. Tokio algoritmo vykdymas pradedamas nuo konkretaus, neoptimalaus uždavinio sprendinio, kuris paprastai būna pasirenkamas atsitiktinai arba remiantis kitu algoritmu, kuris gali sukonstruoti sprendinį nuo nulio. Lokaliai paieška būtini sprendinių kaimynystės ir tikslo funkcijos apibrėžimai. Sprendinių kaimynystė, tai aibė užduoties sprendinių, kuriuos galima gauti atlikus, uždavinio kontekste apibrėžtas, nesudėtingas transformacijas einamajam lokalios paieškos sprendiniui. Grafų redagavimo atstumo kontekste, lokalia paieška grafams G ir H galima formuluoti, kaip optimalaus grafų viršūnių atvaizdžio $\pi^* \in \Pi(G, H)$ paiešką. Tokiame uždavinyje sprendinio transformacijos atitinka viršūnių atvaizdžio π modifikacija, kur vienas ar keli keitiniai $(u_i, v_j) \in \pi$ yra keičiami kitais viršūnių keitiniais (u'_i, v'_j). Tikslo funkcija lokalioje paieškoje yra funkcija, kuri duotam sprendiniui gražiną skaitinę vertę. Lokalios paieškos algoritmų tikslas minimizuoti šią funkciją. Grafų redagavimo atstumo kontekste, tikslo funkcija yra esamo viršūnių atvaizdžio π indukuotos redagavimo sekos P_π kaina $c(P_\pi)$, kur c yra grafo redagavimo operacijų kainų funkcija.

Bendru atveju, lokalios paieškos algoritmas einamam sprendiniui randa visus galimus kaimyninius sprendinius, tada taikant tikslo funkcija įvertina visus galimus kaimyninius sprendinius ir geriausias kandidatas yra pasirenkamas, kaip naujas einamasis lokalios paieškos sprendinys. Toliau algoritmas iteruoja - naujam einamam sprendiniui ieško kaimyninių sprendinių ir sprendžia, kurie iš jų geresni už dabartinį, jeigu tokių nėra, vadinasi rastas lokalus minimumas. Grafų redagavimo atstumo paieškos variante, kiekvienoje algoritmo iteracijoje modifikuojamas grafo viršūnių atvaizdis $\pi \in \Pi(G, H)$ į viršūnių atvaizdį $\pi' \in \Pi(G, H)$, taip kad naujo viršūnių atvaizdžio indukuota

redagavimo seka būtų mažesnė ar lygi einamajai $c(P_{\pi'}) \leq c(P_\pi)$.

Lokalios paieškos algoritmų privalumas tas, kad vykdant paieška yra saugomas einamasis geriausias sprendinys, todėl paieškos laiką ar iteracijų skaičių galima laisvai kontroliuoti ar bet kada nutraukti paiešką. Tačiau lokalios paieškos algoritmų rezultatai stipriai priklauso nuo pasirinkto pradinio sprendinio. Jeigu pasirinktas pradinis sprendinys yra arti lokalaus minimumo, lokalios paieškos algoritmas galimai sustos ties tuo lokaliu minimumu, kuris gali būti smarkiai nutolęs nuo globalaus minimumo. Žinoma, galima įvairiais būdais spręsti šia bėdą, pavyzdžiui vykdant keletą lokalių paieškų iš skirtingų pradinių sprendinių arba aptikus lokalų minimumą mutuoti esamą sprendinį - atsitiktinai pasirinkti kaimyninį sprendinį neatsižvelgiant į tikslo funkcija ar kitaip transformuoti esamą sprendinį siekiant padėti algoritmui ištrūkti iš lokalaus minimumo duobės.

2.3.2. Algoritmų šablonas

Euristinius algoritmus, kurie remiasi lokalios paieškos algoritmais grafų redagavimo atstumui skaičiuoti, galima apibendrinti vienu abstrakčiu algoritmų šablonu, kuris pateiktas iliustracijoje Algoritmas 3. Pirmame žingsnyje, grafams G ir H yra konstruojamas pradinis grafų viršūnių atvaizdis $\pi \in \Pi(G, H)$. Viršūnių atvaizdis π gali būti konstruojamas atsitiktinai arba pasiremiant paprograme iš TSPKT-GRA klasės euristinių algoritmų uždavinio C konstravimui (Algoritmas 1, eilutė 2). Toliau lokalios paieškos algoritmas yra pritaikomas surasti naują grafo viršūnių atvaizdį $\pi' \in \Pi(G, H)$ tokį kad jo indukuota redagavimo sekos kainą būtų mažesnė už pradinio atvaizdžio π indukuotos redagavimo sekos kainą: $c(P_{\pi'}) \leq c(P_\pi)$. Galiausiai trivialiai paskaičiuojama gauto grafo viršūnių atvaizdžio π' indukuotos redagavimo sekos kaina $c(P_{\pi'})$ ir ji gražinama, kaip viršutinis grafų redagavimo atstumo rėžis. Pradinio grafo viršūnių atvaizdžio π, lokalios paieškos pagalba, transformavimas į galutinį π' yra esminė šių algoritmų dalis ir čia skirtingi euristiniai algoritmai labiausiai išsiskiria.

Kadangi lokalios paieškos algoritmai paprastai gražina tik lokalius minimumus, pradinis viršūnių atvaizdis π, nuo kurio yra pradedama paieška, yra labai svarbus. Geros pradžios pasirinkimo problema yra pagrindinis lokalia paieška paremtų euristikų trūkumas. Dėl šio trūkumo egzistuoja keletas algoritmų praplėtimų - algoritmai, kurie tik praplečia egzistuojančius lokalios paieškos euristinius algoritmus ar naudoja juos kaip paprogrames. Pavyzdžiui *MultiStart* [DBG+18] algoritmas praplečia bendrą algoritmų šabloną Algoritmas 3 ir siūlo naudoti k skirtingų pradinių grafo viršūnių atvaizdžių $\pi \in \Pi(G, H)$ ir vykdyti paiešką paraleliai, gražinant geriausią iš k skirtingų paieškos rezultatų. Kita pavyzdį galima imti iš algoritmo *Randpost* [BBB18]. Čia taip pat siūloma vykdyti bendrą algoritmų šabloną Algoritmas 3 k kartų, tačiau konstruojant pradinį grafo viršūnių atvaizdį $\pi_{i+1} \in \Pi(G, H)$ atsižvelgti į prieš tai vykdytos paieškos rezultatą $\pi'_i \in \Pi(G, H)$, kur $i \in (1, 2, \ldots, k-1)$. Tokiu būdu siekiama sukonstruoti geresnį pradžios tašką prieš pradedant naują paiešką.

Algoritmas 3 LP-GRA euristikų šablonas

1: **procedure** LP-GRA(G, H, c_V, c_E)
2: Sukonstruojamas pradinis grafo viršūnių atvaizdis $\pi \in \Pi(G, H)$
3: Naudojantis pradine įvestimi ir lokalios paieškos pagalba sukonstruojamas naujas grafo viršūnių atvaizdis $\pi' \in \Pi(G, H)$
4: $VR \leftarrow c(P_{\pi'})$ ▷ Priskiriamas viršutinis GRA rėžis
5: **return** VR
6: **end procedure**

2.4. Svarbiausi algoritmai

Svarbiausi šių dienų euristiniai grafų redagavimo atstumo paieškos algoritmai apžvelgiami D. B. Blumenthal darbe [BBG+20]. Minėtame darbe yra apžvelgiama trisdešimt skirtingų euristinių algoritmų bei atliekamas visų šių algoritmų empirinis tyrimas vientisoje aplinkoje. Visi algoritmai yra įgyvendinti naudojantis C++ programavimo kalba ir naudojantis tomis pačiomis bazinėmis, darbui su grafais skirtomis, duomenų struktūromis. Darbe atliktų eksperimentų ir pačių algoritmų įgyvendinimas prieinamas atviro kodo bibliotekoje GEDLIB [BBG+19]. Šioje bibliotekoje pateiktas programinis kodas susikompiliuoja ir galima nesudėtingai atkartoti išsamius eksperimentus atliktus originaliame darbe [BBG+20]. Tai puikiai tinkantis įrankis šio darbo tikslui siekti, kadangi esamų euristinių algoritmų modifikavimas ir rezultatų palyginimas su originaliais algoritmais vyks toje pačioje aplinkoje - galima tiksliau įvertinti atliktų modifikacijų poveikį.

Žinoma ne visi trisdešimt euristinių algoritmų yra verti dėmesio ir detalesnio gilinimosi - iš atliktų eksperimentų galima nesunkiai pastebėti, kad kai kurie euristiniai algoritmai nepranoksta kitų algoritmų, nei savo vykdymo laiku, nei rezultatų tikslumu. Šio darbo kontekste įdomūs euristiniai algoritmai, kurie gražina tiksliausius grafų redagavimo atstumo rėžius. Kadangi šiai dienai geriausi euristiniai algoritmai gražina reikšmes, kurios vidutiniškai nutolusios nuo tikslios grafų redagavimo atstumo reikšmės per 2%, yra mažai vietos euristiniu algoritmų tobulinime siekiant tikslesnių rezultatų [BBG+20]. Tačiau spragų algoritmuose, atsižvelgiant į vykdymo laiką, tikrai yra.

Tiksliausias grafų redagavimo atstumo apatinį rėžį gražinantys euristiniai algoritmai yra *ADJ-IP* [JH06] ir *F2* [LAR+17] priklausančių TP-GRA algoritmų klasei, kartu su algoritmu *Branch-Tight* [BG17b], kuris priklauso maišytų algoritmų klasei. Šie algoritmai gražina tiksliausią apatinį rėžį, tačiau skaičiavimus atlieka žymiai lėčiau nei kiti euristiniai algoritmai. Kita vertus, algoritmai *Node* [JH06], *Branch-Fast* [ZZL+14] ir *Branch-Const* [BG17b], priklausantys TSPKT-GRA algoritmų klasei, veikia stebėtinai greičiau ir gražina tik šiek prastesnius rezultatus.

Viršutinio rėžio skaičiavimuose geriausius rezultatus gražina, LP-GRA algoritmų klasei priklausantis, algoritmas *IPFP* [BDB+18]. Didžioji dalis TSPKT-GRA klasės euristinių algoritmų veikia greičiau nei *IPFP*, pavyzdžiui prieš tai minėti *Node*, *Branch-Fast*, *Branch-Const* ar kitos TSPKT-GRA klasės euristikos, kaip *BP* [RB09] ar *Star* [ZTW+09]. Analogiškai, kaip apatinio rėžio skaičiavimuose, tiksliausi algoritmai keliomis eilėmis atsilieka vykdymo laiku lyginant su išvardintais TSPKT-GRA klasės euristiniais algoritmais.

3. Grafai

Šio darbo eksperimentuose atliekami testai įvairiems grafų rinkiniams. Dalis grafų rinkinių atitinka realaus pasaulio esybes, kita dalis grafų buvo sugeneruota. Realaus pasaulio grafų rinkiniai yra prieinami "IAM" grafų rinkinių repozitorijoje [RB08]. Šiame darbe pasirinkti šeši skirtingi neorientuotų grafų rinkiniai, kurių viršūnių bei briaunų statistika pavaizduota lentelėje 2. *AIDS* ir *MUTA* grafų rinkiniai reprezentuoja molekulinius junginius. Molekulės iš grafų rinkinio *AIDS* skirstomos į molekulių klasę, kuri pasireiškia ir molekulių klasę, kuri nepasireiškia aktyvumu ties ŽIV. Panašiai, molekulės iš grafų rinkinio *MUTA* skirstomos į molekulių klasę, kurios sukelia ir molekulių klasę, kurios nesukelia genetinių mutacijų. Grafai iš rinkinio *PROTEIN* reprezentuoja šešių skirtingų klasių baltymus. Rinkinio *GREC* grafai reprezentuoja įvairius architektūros ir elektronikos brėžinių simbolius, kurie patyrė tam tikro lygio iškraipymą. Šio rinkinio grafai klasifikuojami į 22 skirtingas klases. Rinkinį *LETTER* sudaro grafai, kurie reprezentuoja iškraipytas rašytines lotynų raides. Galiausiai grafai iš rinkinio *FINGER* reprezentuoja pirštų antspaudus - šio rinkinio grafai nėra dideli kadangi grafuose atvaizduojami tik svarbiausi antspaudo regionai.

Rinkinys	**Viršūnių sk.**			**Briaunų sk.**		
	Min.	Maks.	Vid.	Min.	Maks.	Vid.
AIDS	7	78	22	6	88	23.2
MUTA	6	176	33.1	5	106	32.7
PROTEIN	7	96	31.8	14	121	60.9
GREC	4	24	11.7	2	29	12.2
LETTER	2	8	4.7	1	8	4.48
FINGER	2	19	7.18	1	17	5.74

2 lentelė. Eksperimentuose naudotų realaus pasaulio grafų viršūnių ir briaunų kiekio statistika.

Lentelėje 3 yra pavaizduota platesnė realaus pasaulio grafų rinkinių statistika. Lentelėje pateikiamos reikšmės yra visų rinkinio grafų vidurkis duotai statistikai, tai yra viršūnių laipsnio, briaunų tankio, skersmens ir spindulio reikšmėms. Atkreipus dėmesį į minimalų viršūnių laipsnį rinkiniuose *AIDS* bei *MUTA* pastebime, kad reikšmė yra mažesnė už 1. Tai reiškia, kad dalis grafų yra sudaryti iš vienos viršūnės arba dalis grafų yra nejungūs ir turi pavienių viršūnių, kurios neturi briaunų. Kaip matoma iš lentelės 2, visi rinkinių *AIDS* ir *MUTA* grafai turi bent 6 viršūnes, kas reiškia, kad dalis grafų yra nejungūs. Taip pat galima pastebėti, kad rinkinyje *PROTEIN*, minimalus ir maksimalus viršūnių laipsnis yra ženkliai aukštesnis už likusių rinkinių laipsnius. Rinkinių *AIDS*, *MUTA* ir *PROTEIN* vidutinė spindulio ir skersmens statistika beveik identiška, tačiau vidutinis viršūnių ir briaunų skaičius skiriasi.

3.1. Laipsninio dėsnio grafai

Laipsninis dėsnis (angl. power law) yra funkcinis ryšys tarp dviejų kiekybių, kur pokytis vienai kiekybei yra proporcingas kitos kiekybės pokyčiui. Šis dėsnis užrašomas formule $f(x) = x^k$, kur k yra laipsnis nusakantis ryšį tarp kiekybių. Įvairios sistemos, kaip genetiniai tinklai ar pasaulinis

Rinkinys	Viršūnių laipsniai			Skersmuo	Spindulys	Briaunų tankis
	Min	Maks.	Vid.			
AIDS	0.97	3.33	2.07	10.9	5.77	0.148
MUTA	0.94	3.81	2.03	10.3	5.49	0.087
PROTEIN	2.49	6.02	3.89	11.2	5.88	0.163
GREC	1.23	2.89	2.01	4.91	3.11	0.231
LETTER	1.24	2.58	1.88	2.58	1.63	0.582
FINGER	1	1.81	1.49	4.45	2.51	0.384

3 lentelė. Eksperimentuose naudotų realaus pasaulio grafų statistika.

tinklas, yra apibūdinami, kaip tinklai su sudėtinga grafo topologija. Bendras bruožas siejantis didelio masto realaus pasaulio tinklus yra toks, kad viršūnių jungumas seka laipsninio dėsnio pasiskirstymą. Tai yra, tikimybė $P(k)$ viršūnei v turėti briaunas su k kitų viršūnių mažėja pagal laipsninį dėsnį $P(k) = k^{-\gamma}$. Šis bruožas yra pasekmė dviejų svarbių veiksnių besiformuojant tinklams: naujų viršūnių atsiradimas ir pirmenybinio (angl. preferential) viršūnių prisijungimo prie jau susiformavusių ir tankiai sujungtų klasterių [BA99].

3.1.1. Grafo modelis

Barabasio ir Alberto darbe [BA99] pristatytas laipsninį dėsnį atitinkantis grafo konstravimo modelis, kuris remiasi grafo augimo bei pirmenybinio viršūnių prijungimo principais. Dauguma realaus pasaulio tinklų yra atviri ir formuojasi augant grafo viršūnių skaičiui. Kaip pavyzdys: aktorių tinklas, kurį galima vaizduoti grafu, kur viršūnės atitinka aktorius, o briaunos jungiančios jas reprezentuoja atvejus, kai du aktoriai vaidino tame pačiame filme. Atsiradus naujam aktoriui sistemoje, tikėtina, kad jis užims pagalbinio aktoriaus vaidmenį filmuose, kur kartu vaidins jau patyrę ir labiau žinomi pirmo plano aktoriai. Atitinkami, tikimybė naujam aktoriui vaidinti filme su patyrusiu aktoriumi didesnė, nei su kitu, mažiau patyrusiu ir žinomu aktoriumi. Šis pavyzdys rodo, kad tikimybė, su kuria nauja viršūnė prisijungia prie jau egzistuojančio grafo nėra universali, bet turi didesnę tikimybę prisijungti prie viršūnių, kurios jau turi daug jungčių su kitomis viršūnėmis.

Dėl pirmenybinio viršūnių prijungimo, viršūnės, kuri turi daugiau briaunų nei kita viršūnė, laipsnis augs spartesniu tempu, nei kitos viršūnės. Tokiu būdu augančio grafo tikimybė $P(k)$ viršūnei v turėti briaunas su k kitų viršūnių mažėja pagal laipsninį dėsnį $P(k) = k^{-\gamma}$, kur eksponentė $\gamma = 2.9 \pm 0.1$ [BA99].

3.1.2. Grafo generavimas

Eksperimentams su laipsninio dėsnio grafais atlikti buvo implementuotas algoritmas jų generavimui [Par11]. Šio modelio grafų generavimo procese, kiekvieno laiko žingsnyje yra pridedama nauja viršūnė su m briaunų. Tikimybė, kad nauja viršūnė v bus sujungta su jau egzistuojančia grafo viršūne u yra proporcinga viršūnės laipsniui $d(u)$. Šio algoritmo implementacijos pseudokodas pateiktas priede 4. Grafas su n viršūnių $\{1, 2, \ldots, n\}$ yra kuriamas pirma užpildant grafo

viršūnių aibę V ir grafo briaunų aibę E pradiniais elementais. Pirma, aibė V užpildoma n_0 pradinių viršūnių, kur $n_0 = m+2$, aibė E užpildoma briaunomis sukuriant ciklą tarp pradinių grafo viršūnių, masyvas PS užpildomas viršūnių laipsnių sumomis. Toliau kiekvienos grafo auginimo iteracijos metu yra pridedama nauja viršūnė v_i, kur $i \in \{n_0, n_0+1, \ldots, n\}$, ir m naujų briaunų jungiančių v_i su jau egzistuojančiomis grafo viršūnėmis. Šios iteracijos kartojamos $n - n_0$ kartų, po kurių yra gaunamas grafas su n viršūnių. Naujos viršūnės generavimo iteracijos metu masyvo elementas $PS[v_i]$ saugo reikšmę $\sum_{k=1}^{v_i} d[k]$ kiekvienai, iteracijos laiko momentu, egzistuojančiai grafo viršūnei. Procedūra *RandomNumber*$(1, PS[i{-}1]) \rightarrow r$ gražina atsitiktini skaičių r iš intervalo $(1, \sum_{k=1}^{v_{i-1}} d[k])$, kuris toliau perduodamas procedūrai *FindNode*$(r, PS) \rightarrow v_j$, kuri parenka atitinkama grafo viršūnę. Jei tarp einamos grafo viršūnės v_i ir procedūros *FindNode* parinktos viršūnės v_j neegzistuoja briauna, ji yra įterpiama į grafą, kitu atveju atsitiktinis skaičius r yra generuojamas iš naujo tol kol randama viršūnė v_j, kuriai galima įterpti briauną. Tokiu būdu, viršūnei v_i iš viso įterpiamos m briaunos.

3.2. Klasterizuoti grafai

Šiame darbe klasterizuotais grafais įvardinami grafai, kurie pasižymi dideliu klasterizavimo koeficientu – tikimybe, kad grafo viršūnės v_1 kaimyninės viršūnės v_2 ir v_3 taip pat yra kaimynės viena kitai, kitaip sakant, visos trys viršūnės yra sujungtos briaunomis sudarant trikampį. Didžioji dalis realaus pasaulio grafų, ypatingai socialinių tinklų, pasižymi aukštu klasterizavimo koeficientu [BA99]. Tokių grafų viršūnės sudaro klasterius, kuriuose viršūnės tarpusavyje yra tankiai sujungtos briaunomis, tuo tarpu viršūnės tarp skirtingų klasterių turi mažesne tikimybę jungtis briaunomis. Toliau aprašoma klasterizuotų grafų generavimo procedūrą.

3.2.1. Klasterizuotų grafų generavimas

Generuojant klasterizuotus grafus yra parenkamas viršūnių kiekis n, klasterių kiekis c, tikimybė briaunai viduje klasterio P_c ir tikimybė briaunai tarp klasterių P_i. Pirma sukuriamas grafas iš n viršūnių ir sukuriamas ciklas tarp viršūnių, taip užtikrinant grafo jungumą. Kiekviena grafo viršūnė $n_i \in \{1, 2, \ldots, n\}$ yra priskiriama klasteriui c_j, kur $j = \lfloor \frac{i}{c} \rfloor + 1$. Viršūnių skaičius klasteriuose neviršija $\lfloor \frac{n}{c} \rfloor$, o atveju kai $n \mod c \neq 0$, sukuriamas papildomas klasteris c', kuriam priskiriamos viršūnės n_i, kur $i > \lfloor \frac{n}{c} \rfloor \cdot c$. Toliau kiekvienai galimai grafo briaunai yra generuojamas atsitiktinis įvykis ir pritaikoma briaunos egzistavimo tikimybė P_c arba P_i. Jei einamoji briauna yra tarp viršūnių, kurios priklauso tam pačiam klasteriui – pritaikoma tikimybė P_c. Analogiškai, jei briauna yra tarp viršūnių, kurios priklauso skirtingiems klasteriams – pritaikoma tikimybė P_i.

Siekiant gauti grafą, kuris panašus į realiame pasaulyje sutinkamus klasterizuotus grafus, reikėtų tinkamai pasirinkti tikimybinius parametrus P_c ir P_i, su būtina nelygybė $P_c > P_i$. Šiame darbe naudojami rėžiai $0.75 \leq P_c \leq 1$ ir $0 \leq P_i \leq 0.2$.

4. Principiniai sprendimai

Šio darbo tikslas patobulinti geriausius grafo redagavimo atstumo paieškos euristinius algoritmus, siekiant paspartinti jų vykdymo laiką ar pagerinti gražinamų aproksimacijų tikslumą. Euristinius GRA algoritmus galima sugrupuoti į dvi grupes: algoritmus, kurie gražina apatinį rėžį ir algoritmus, kurie gražina viršutinį rėžį. Tiesa, kai kurie algoritmai, apskaičiuoja abu rėžius iš karto, tačiau nėra algoritmo, kuris vienu metu gražintu tiksliausius apatinį ir viršutinį rėžius [BBG+20]. Atsižvelgiant į tai, kad euristikų, kurios gražina viršutinį GRA rėžį yra žymiai daugiau nei euristikų, kurios gražina apatinį rėžį, šiame darbe fokusuojamasi į būtent šiuos algoritmus ir buvo spartinamas tiksliausias šiai dienai žinomas algoritmas *IPFP* [BBG+20].

IPFP yra iteratyvaus Frank-Wolf algoritmo [FW+56] adaptacija grafų redagavimo atstumui aproksimuoti. Šis algoritmas pradeda savo darbą nuo pradinio grafų viršūnių atvaizdžio $\pi_0 \in \Pi(G, H)$ ir toliau iteruodamas konverguoja iki, galimai trupmeninio, lokalaus minimumo grafų redagavimo atstumui.

Šis algoritmas pradeda savo darbą inicializuodamas redagavimo atstumo viršutinį rėžį $VR := c(\pi_0)$, kur c yra redagavimo kainų funkcija ir π_0 yra pradinis grafų viršūnių atvaizdis perduotas algoritmui, kaip įvestis. Toliau algoritmas iteruoja ir kiekviena iteracija pradedama nuo tiesinės sumos priskyrimo su klaidų taisymu (toliau TSPKT) uždavinio C_k konstravimo remiantis grafų viršūnių atvaizdžiu π_k. Toliau TSPKT uždavinys yra optimaliai išsprendžiamas ir gaunamas klaidas taisantis derinys X^*_{k+1}, kurio dėka yra atnaujinamas viršutinis rėžis $VR := min(VR, c(X^*_{k+1}))$. Toliau algoritmas apskaičiuoja žingsnį $\alpha_{k+1} \in [0, 1]$, kurio pagalba sukonstruoja naują, galimai trupmeninį, grafų viršūnių atvaizdį π_{k+1} ir pradeda sekančia iteracija. Algoritmas iteruoja tol, kol pasiekia maksimalų leistina iteracijų skaičių I arba tol kol einamasis uždavinys neperžengia konvergavimo slenksčio ϵ, kitaip sakant algoritmas sustoja, kai iteracijos gautas viršutinis rėžis beveik nesiskiria nuo prieš tai buvusios iteracijos gauto rėžio. Galiausiai, galimai trupmeninis grafų viršūnių atvaizdis π_{k+1} yra projektuojamas į sveiką viršūnių atspindį $\hat{\pi}$ ir gaunamas galutinis algoritmo rezultatas: $VR := min(VR, c(\hat{\pi}))$.

4.1. Algoritmo IPFP spartinimas

Algoritmo *IPFP* vykdymo laikas priklauso nuo to kiek lokalios paieškos iteracijų yra įvykdoma prieš algoritmui sustojant. Kadangi algoritmas iteruodamas konverguoja iki lokalaus minimumo, iteracijų skaičių lemia pradinio paieškos taško parinkimas - pradinis grafų viršūnių atvaizdis $\pi_0 \in \Pi(G, H)$. Kuo pradinis viršūnių atvaizdis yra arčiau lokalaus grafų redagavimo atstumo minimumo, tuo algoritmas greičiau konverguos ir baigs savo darbą. Nustatyti, ar duotas grafų viršūnių atvaizdis π yra arti lokalaus minimumo, nėra paprasta, kadangi tai reikalauja ištirti GRA paieškos erdvę aplink duotą pradinį tašką, o tai nėra mažos apimties erdvė. Kita vertus, šio darbo eksperimentuose pastebėta, kad kuo grafų viršūnių atspindžio $\pi \in \Pi(G, H)$ indukuojamos grafų redagavimo sekos kaina $c(\pi)$ yra mažesnė, tuo viršūnių atvaizdis yra arčiau lokalaus GRA minimumo ir *IPFP* konverguos greičiau. Tačiau, kaip ir kitose kompiuterių mokslo srityse, algoritmo vykdymo laikas nėra vienintelis svarbus kintamasis - svarbus ir algoritmo išgaunamų rezultatų

tikslumas.

Grafų redagavimo atstumo kontekste, viena iš platesnių šio mato euristikų panaudojimo sritis yra modelių atpažinimas (angl. pattern recognition). Yra parodyta, kad modelių atpažinimo karkasai veikia geriau, jeigu juose naudojamos GRA euristikos gražina tikslesnes GRA aproksimacijas [BBG+20], kitaip sakant išgauna rėžius, kurie yra, kaip įmanoma arčiau tikros GRA reikšmės. Kalbant apie GRA viršutinio rėžio skaičiavimą naudojantis lokalia paieška, intuityviai, galima galvoti, kad kuo pradinė viršūnių atspindžio π_0 redagavimo kainą yra mažesnė, tuo lokali paieška išgaus geresnius rezultatus - geresnė pradžia, geresnis galutinis rezultatas. Tačiau šiame darbe yra parodoma, kad tai netiesa algoritmui *IPFP*. Anaiptol viršūnių atvaizdis π_x su maža redagavimo kaina gali nuvesti iki prastesnio galutinio sprendinio, negu atsitiktinai parinktas viršūnių atvaizdis π_y su didesne redagavimo kaina, kitaip sakant: $c(\pi_x) < c(\pi_y) \not\Rightarrow c(IPFP(\pi_x)) < c(IPFP(\pi_y))$. Šį dėsningumą galima pastebėti bandant taikyti pradinį viršūnių atspindį gautą iš kito lokalios paieškos algoritmo, pavyzdžiui *Refine* [BBB+20]. Šis lokalios paieškos algoritmas yra greitesnis už *IPFP* tačiau ne toks tikslus. Taikant šio algoritmo rastus viršūnių atspindžius dideliems grafams, *IPFP* gražina prastesnius rezultatus, negu pradedant nuo viršūnių atspindžių gražintų, ne tokių tikslių, TSPKT-GRA klasės algoritmų.

Gero pradinio sprendinio, grafų viršūnių atspindžio π_0 parinkimas, yra svarbi algoritmo *IPFP* dalis. Deja klausimas, kaip parinkti pradinį sprendinį nėra trivialus ir atsakymas priklauso nuo uždavinio konteksto - grafų dydžio, tankio ir jų struktūros. Originaliame algoritmą *IPFP* pristačiusiame darbe [BGB16], pradinio sprendinio parinkimas nėra analizuojamas ir eksperimentuose naudojamas TSPKT-GRA klasės algoritmas *Walks* [GBR+14]. Darbe, kuris atliko trisdešimties skirtingų GRA euristinių algoritmų empirinį tyrime, algoritmas *IPFP* buvo testuojamas su atsitiktinai parinktais pradiniais sprendiniais.

Šiame darbe yra nagrinėjami skirtingi algoritmai siekiant nustatyti kada ir koks algoritmas geriausiai tinka pradiniam grafų viršūnių atspindžiui π_0 parinkti. Yra parodoma, kurios GRA euristikos geriausiai veikia mažuose grafuose, o kurias geriau taikyti dideliems grafams. Taip pat įvertinamas algoritmo *IPFP* vykdymo laikas pradedant nuo skirtingais algoritmais išgautų viršūnių atvaizdžių. Taip pat šiame darbe pristatomos kelios TSPKT-GRA klasės algoritmų modifikacijos, kurių pradiniai sprendiniai didesniuose grafuose privedė algoritmą *IPFP* iki tiksliausių GRA rėžių. Atsižvelgiant į šiuos rezultatus yra daroma prielaida, kad yra vietos tolimesniam pradinių grafų viršūnių atspindžių tobulinimui.

4.2. TSPKT-GRA klasės euristikų modifikacijos

TSPKT-GRA algoritmų svarbiausia dalis yra uždavinio $C \in \mathbb{R}^{(|V_G|+1)\times(|V_H|+1)}$ konstravimas - būtent šiame žingsnyje skirtingos šios klasės euristikos išsiskiria viena nuo kitos. Matrica C reprezentuoja apytiksles viršūnių iš grafo G keitimo į viršūnes iš grafo H kainas. Čia svarbu prisiminti, kad grafų redagavimo atstumą sudaro ne tik viršūnių modifikacijos, bet ir briaunų modifikacijos. Atlikus viršūnės trynimo operacija, netiesiogiai ištrinamos ir visos su ta viršūne susijusios briaunos. Atlikus viršūnių keitinius $v_u^G \to v_j^H$ ir $v_v^G \to v_k^H$, netiesiogiai indukuojama ir briaunos tarp šių viršūnių trynimo, pridėjimo ar modifikacijos kaina. Dėl to, konstruojant už-

davinį C reikia atsižvelgti į galimas briaunų modifikacijų redagavimo kainas. Kadangi briaunų redagavimo kainos priklauso nuo to kokios dvi viršūnės bus keičiamos kitomis dvejomis, o visų galimų viršūnių atvaizdžių skaičius $|\Pi(G,H)|$ auga eksponentiškai - viršūnių keitimo kainas matricoje C galima tik aproksimuoti. Sukonstravus tiesinių sumų priskyrimo uždavinį C, jis yra optimaliai išsprendžiamas.

Šiame darbe pristatomos dešimt TSPKT-GRA klasės algoritmų modifikacijų. Nors daugelis išbandytų modifikacijų nedavė gerų rezultatų, kelioms modifikacijos pavyko išgauti geresnius GRA skaičiavimo rezultatus - šių modifikacijų sukonstruoti grafų viršūnių atvaizdžiai $\pi \in \Pi(G,H)$, algoritmo *IPFP* pagalba konvergavo į tiksliausius rezultatus vidutinio dydžio grafuose.

4.2.1. Algoritmo Star modifikacijos

Originalus *Star* algoritmas pildydamas uždavinį C kiekvienam viršūnių keitiniui paskaičiuoja kiek briaunų bus paveikta šio keitimo. Viršūnių pridėjimo $\epsilon \to v_j$ ir ištrynimo atveju $v_i \to \epsilon$ algoritmas gražina viršūnės laipsnį. Viršūnių pervadinimo atveju $v_i \to v_j$ algoritmas paskaičiuoja kiek briaunų dalinasi duotos viršūnės ir gražina skirtumą. Galiausiai algoritmas matricą C padaugina iš konstantos c_{min}, kuri lygi mažiausiai visų galimų grafų koregavimo kainų iš duotos kainų funkcijos c.

Šio algoritmo pagrindu buvo atliktos septynios modifikacijos. Algoritmas *Star-2* yra algoritmų *Star* ir *BP* kombinacija, kur viršūnių keitimo kainos matricoje C yra pildomos *Star* algoritmu, o viršūnių pridėjimo bei ištrynimo kainos yra apskaičiuojamos naudojantis metodu iš algoritmo *BP*. Visos viršūnių keitinių aproksimacijos yra padauginamos iš konstantos c_{min}, kaip ir originaliame *Star* algoritme. Algoritmai *BP*, *Branch* ir *Branch-Fast* naudoja tą patį metodą viršūnių pridėjimo ir ištrynimo kainoms aproksimuoti. Modifikacija *Star-3* yra beveik identiška modifikacijai *Star-2*, tiesiog viršūnių pridėjimo ir ištrynimo kainos nėra dauginamos iš konstantos c_{min}. Modifikacija *Star-4* yra labai paprasta - vietoj konstantos c_{min} yra naudojamos trys konstantos: c^S_{avg}, c^I_{avg} ir c^R_{avg} - tai yra atskirų modifikavimo operacijų, viršūnių keitimo, pridėjimo ir ištrynimo, vidutinės koregavimo kainos viršūnėms ir briaunoms. Šios konstantos yra dauginamos atskiroms matricos C dalims, tai yra viršūnių keitimo matricos dalis C^S yra dauginama iš c^S_{avg}, viršūnių pridėjimo dalis C^I iš c^I_{avg} ir viršūnių trynimo dalis C^R iš c^R_{avg}. Algoritmo modifikacija *Star-5* įtraukia *Star-4* atliktas modifikacijas ir prideda papildoma logika aproksimuojant viršūnių pridėjimo ir ištrynimo kainas. Jeigu duoti grafai G ir H yra vienodo dydžio, tai yra turi vienodą skaičių viršūnių $|V_G| = |V_E|$, tai viršūnių pridėjimo ir ištrynimo kainos yra užpildomos begalybės reikšmėmis. Modifikacija *Star-6* yra identiška modifikacijai *Star-5* su vienu skirtumu - begalybės reikšmės yra pildomos viršūnių pridėjimo ir ištrynimo keitiniams tik jeigu grafai G ir H yra panašaus dydžio, tai yra $||V_G| - |V_H|| \leq \Delta$, kur $\Delta = \lceil \frac{min(|V_G|,|V_H|)}{10} \rceil$. Jeigu $|V_G| \leq |V_H|$, tai uždavinio C viršūnių trynimo kainų dalis C^R yra užpildoma begalybės reikšmėmis. Jeigu $|V_G| \geq |V_H|$, tai uždavinio C viršūnių pridėjimo kainų dalis C^I yra užpildoma begalybės reikšmėmis. Modifikacija *Star-7*, panašiai kaip *Star-3*, yra algoritmų *Star* ir *Node* kombinacija, kur viršūnių keitimo kainos matricoje C yra apskaičiuojamos algoritmo *Star* metodais, o viršūnių pridėjimo bei ištry-

nimo kainos yra tiesiog paimamos iš viršūnių redagavimo kainų funkcijos c_V ir neatliekami jokie papildomi euristiniai skaičiavimai, kaip kituose TSPKT-GRA algoritmuose. Galiausiai, modifikacija *Star-8* yra panaši į modifikaciją *Star-7* - viršūnių pervadinimo kainos, arba kitaip pasakius matricos dalis C^S, yra užpildoma reikšmėmis tiesiai iš redagavimo kainų funkcijos c_V, o viršūnių pridėjimo ir ištrynimo kainos yra apskaičiuojamos originalaus algoritmo *Star* metodais.

4.2.2. Algoritmo Branch-Const modifikacijos

Likusi dalis TSPKT-GRA algoritmų modifikacijų yra paremtos algoritmų *Branch-Const*. Šis algoritmas pildydamas uždavinio matricą C kiekvienam viršūnių keitiniui atsižvelgia ir į galimus briaunų pakeitimus. Viršūnių pridėjimo $\epsilon \to v_j$ ir ištrynimo atveju $v_i \to \epsilon$ algoritmas susumuoja tiesioginę viršūnės keitimo kainą iš funkcijos c_V ir duotos viršūnės laipsnį padaugintą iš mažiausios galimos briaunos redagavimo kainos duotai operacijai. Viršūnių pervadinimo atveju $v_i \to v_j$, algoritmas paskaičiuoja kiek briaunų dalinasi šios viršūnės ir kiek briaunų reikės pridėti arba ištrinti atliekant šį keitinį. Toliau susumuojama viršūnės pervadinimo kaina, bendrų briaunų pervadinimo kaina bei pridedamu arba ištrinamų briaunų kaina.

Algoritmo *Branch-Const* pagrindu buvo atliktos trys modifikacijos. Modifikacija *Branch-Const-2* skiriasi nuo originalaus algoritmo tiktais briaunų redagavimo kainų konstantomis, kurios naudojamos aproksimuojant viršūnių redagavimo kainas. Vietoj mažiausių galimų redagavimo kainų $min\ c(e, f)$ naudojama vidutinė briaunų redagavimo kaina $avg\ c(e, f)$, kur $e \in E_G \cup \{\epsilon\}$ ir $f \in E_H \cup \{\epsilon\}$. Modifikacija *Branch-Const-3*, yra algoritmų *Branch-Const* ir *Node* kombinacija, kur viršūnių keitimo kainos matricoje C yra apskaičiuojamos algoritmo *Branch-Const* metodais, o viršūnių pridėjimo bei ištrynimo kainos yra tiesiog paimamos iš viršūnių redagavimo kainų funkcijos c_V. Modifikacija *Branch-Const-4* yra labai panaši tiktais vietoj viršūnių pridėjimo ir ištrynimo kainų, viršūnių pervadinimo kainos yra užpildomos reikšmėmis tiesiai iš funkcijos c_V.

4.3. Klasikinio mašininio mokymosi algoritmų konstravimas

Klasikinis mašininis mokymasis yra dirbtinio intelekto atšaka. Šios atšakos algoritmai duotai užduočiai ir apmokymo duomenų aibei sukonstruoja modelį, kurį toliau galima taikyti užduoties sprendimui su duomenimis, kurių nebuvo tarp duomenų aibės naudotos modelio konstravimui. Mašininiu mokymusi galima spręsti įvairius uždavinius, tačiau šiame darbe fokusuojamasi ties klasifikavimo uždaviniu.

Klasifikavimas, tai procesas, kurio metu duotam duomenų aibės X įrašui $\bar{x} \in X$ yra priskiriama tam tikra klasė. Duomenų įrašas sudaromas iš aibės skirtingų reikšmių. Toks įrašas apibrėžiamas vektoriumi $\bar{x} = \{x_1, x_2, \ldots, x_n\}$, kur $x_i, i \in \{1, 2, \ldots, n\}$ yra vadinamas atributu - viena iš įrašo aibės reikšmių. Kiekvienas duomenų aibės X įrašas priklauso vienai iš baigtinės aibės Y klasei $y \in Y$ - tai yra tikroji įrašo klasė. Klasifikavimo modelis yra funkcija $M : X \to Y$, kuri įrašus $\bar{x} \in X$ priskiria, kuriai nors klasei $y \in Y$, tačiau parinkta klasė y nebūtinai yra tikroji įrašo klasė. Mašininio mokymosi algoritmų, kurie konstruoja klasifikavimo modelius, tikslas yra sukonstruoti kuo tikslesnį modelį - tai yra modelį, kuris turi didžiausią tikimybę duotam įrašui parinkti jo tikrąją klasę.

Šiame darbe yra konstruojami klasifikavimo modeliai, kuriu klasifikavimo uždavinys yra duotiems grafams G ir H parinkti tiksliausią algoritmą GRA skaičiavimui. Kitaip sakant, šių modelių konstravimo idėja yra atsižvelgti į duotų grafų G ir H struktūrą, panašumus, skirtumus ir kitas grafų savybes, bandant nuspėti, kuris euristinis GRA algoritmas gražins geriausius rezultatus.

4.3.1. Pasirinkimų medžiais paremti algoritmai

Pasirinkimų medžiai yra vienas iš daugelio klasifikavimo algoritmų sukuriamų modelių tipų. Pasirinkimų medis yra beciklis grafas, paprastai binarinis medis, kurio šaknis ir vidinės viršūnės apibrėžia tam tikrą duomenų įrašo atributo požymį, o medžio lapai nusako įrašo klasę. Taip turint įrašą $\bar{x}$, galima visada vienareikšmiškai rasti pasirinkimo medžio lapą, kurio kelias iki medžio šaknies apibūdiną įrašą. Pavyzdžiui, jeigu visi įrašo $\bar{x}$ atributai $x_i \in \bar{x}$ yra skaitinės reikšmės, tai pasirinkimo medžio šaknį ir vidines viršūnes reprezentuos nelygybę $x_i < r$, $r \in \mathbb{R}$.

Siekiant sukonstruoti pasirinkimų medį geriausio GRA algoritmo parinkimui, reikia apibrėžti, kaip atrodys duomenų įrašas bei klasės, kurioms bus naudojamos įrašų klasifikavimui. GRA skaičiavimui, įrašo klasių aibė Y yra sudaryta iš visų šiame darbe nagrinėjamų euristinių GRA algoritmų. Kitaip pasakius, šio klasifikavimo modelio tikslas, parinkti GRA algoritmą, kuris turėtų gražinti geriausius rezultatus. Kadangi GRA yra skaičiuojamas tarp dviejų grafų G ir H, įrašo $\bar{x}$ atributai turėtų atvaizduoti duotų grafų požymius, panašumus ar skirtumus. Šiame darbe apibrėžiama ir naudojama funkcija *GraphDiff*$(G, H) \rightarrow \bar{x}$, kurios sukonstruojamų atributų aibė apibrėžiama lentelėje 4. Atributai pažymėti simboliu Δ yra skaičiuojami pagal formulę $\frac{min(x_i(G), x_i(H))}{max(x_i(G), x_i(H))}$, kur $x_i(G)$ ir $x_i(H)$ žymi tam tikrą grafo savybę, pavyzdžiui viršūnių santykio atribute V_Δ, $x_i(G)$ formulėje atitinką $|V_G|$, spindulių santykio atribute r_Δ, $x_i(G)$ formulėje atitinka $r(G)$, ir taip toliau. Sutampančių viršūnių žymų koeficiento atributas l_V^* apskaičiuojamas susumavus sutampančių žymų kiekį ir padalinus ta skaičių iš didžiausio viršūnių skaičiaus tarp duotų grafų. Analogiškai apskaičiuojamas sutampančių briaunų žymų atributas l_E^*. Paskutinis šiame darbe naudotas atributas j yra būlio tipo, kuris įgyja reikšmę 1 tiktais jei abu duoti grafai yra jungūs. Visų atributų reikšmės patenka į rėžius $0 \leq x_i \leq 1$.

$\bar{x}$	**Atributas**	**Rėžiai**	**Komentaras**
x_1	V_Δ	$0 \leq V_\Delta \leq 1$	Viršūnių santykis
x_2	E_Δ	$0 \leq E_\Delta \leq 1$	Briaunų santykis
x_3	v_Δ^*	$0 \leq v_\Delta^* \leq 1$	Vidutinio viršūnių laipsnių santykis
x_4	t_Δ	$0 \leq t_\Delta \leq 1$	Briaunų tankių santykis
x_5	r_Δ	$0 \leq r_\Delta \leq 1$	Spindulių santykis
x_6	d_Δ	$0 \leq d_\Delta \leq 1$	Skersmenų santykis
x_7	l_V^*	$0 \leq l_V^* \leq 1$	Sutampančių viršūnių žymų koeficientas
x_8	l_E^*	$0 \leq l_E^* \leq 1$	Sutampančių briaunų žymų koeficientas
x_9	j	$j \in \{0, 1\}$	Abu grafai jungūs

4 lentelė. Atributų apibrėžimai, naudoti grafų palyginimui ir pasirinkimų medžių konstravimui.

4.3.2. Sukonstruoti pasirinkimų medžiai

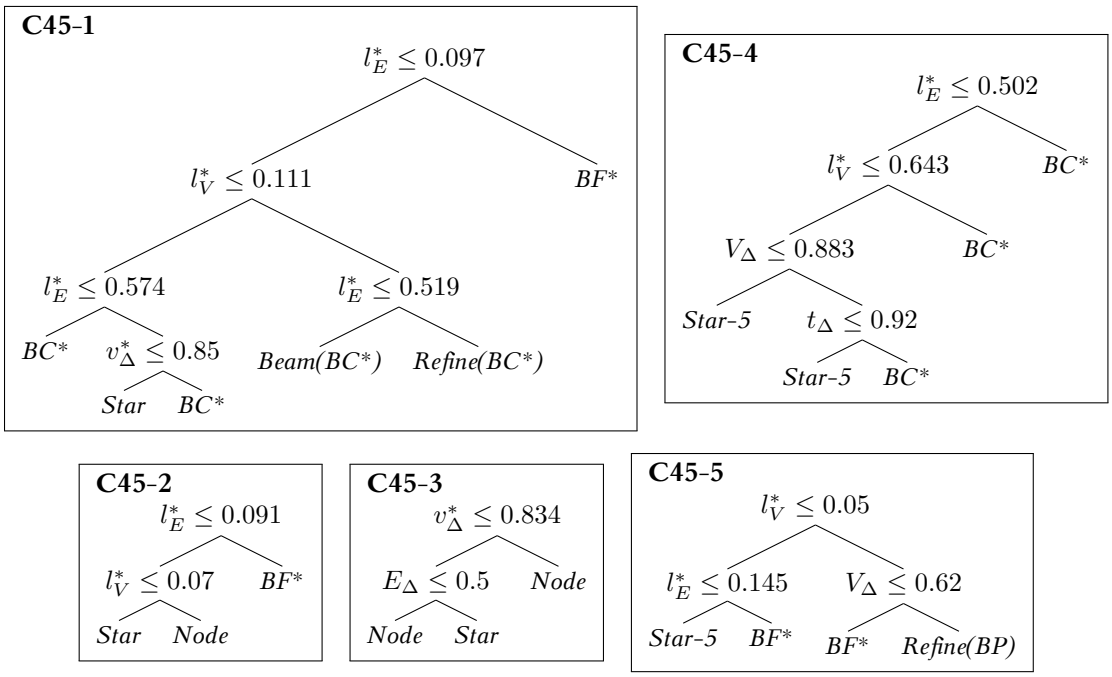
Šiame darbe buvo naudoti du mašininio mokymosi algoritmai pasirinkimų medžiams konstruoti: *C4.5* [Qui14] ir *RepTree* [Kal15]. Abu šie algoritmai implementuoti mašininio mokymosi karkase "*WEKA*" [HFH+09]. "*WEKA*" yra mašininio mokymosi karkasas, kuriame surinkti moderniausi šių laikų mašininio mokymosi algoritmai, duomenų gavybos (angl. data mining) algoritmai, bei įvairūs įrankiai skirti duomenų apdorojimui ir jų analizei.

Lentelėje 5 apibrėžiami pasirinkimų medžių modeliai, kurie buvo sukonstruoti ir analizuoti šiame darbe. Modeliai buvo konstruojami algoritmais *C4.5* ir *RepTree* keičiant naudotas apmokymo įrašų aibes, bei naudojamus atributus. Iliustracijose 1 ir 2 vaizduojami šių algoritmų sukonstruoti pasirinkimų medžiai. Apmokymo grafų rinkiniai yra plačiau aprašomi ketvirtame šio darbo skyriuje. Apmokymo įrašų aibės buvo sugeneruotos vykdant grafų redagavimo atstumo paiešką kiekvienai unikaliai grafų porai iš duoto grafų rinkinio, taikant visus GRA metodus. Vienas įrašas apmokymo aibėje atitinka vieną grafų porą ir GRA metodą, kuris gražino geriausias rezultatus šiai porai, tai yra tiksliausią viršutinį rėžį. Jeigu yra keli metodai, kurie gražino ta patį rėžį, paliekamas metodas su greičiausiu vykdymo laiku. Toliau. kiekvieno įrašo grafams G ir H yra apskaičiuojami atributai, apibrėžti lentelėje 4. Galiausiai, visų įrašų aibė yra skeliama į dvi dalis: dvi trečiosios įrašų yra skiriamos modelio apmokymui ir likęs trečdalis visų įrašų yra skiriami modelio testavimui. Lentelėje 5 pateikiamas sukonstruoto modelio tikslumas. Šis skaičius rodo kiek įrašų iš testavimo aibės, buvo suklasifikuoti teisingai.

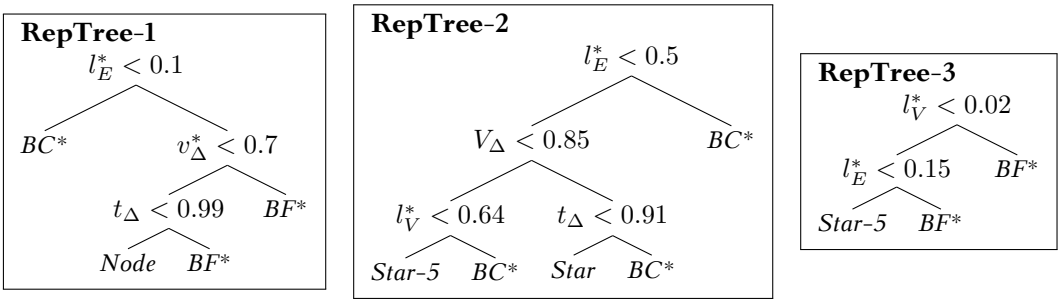
Modelis	**Apmokymo grafų rinkiniai**	**Įrašų sk.**	**Klasių sk.**	**Modelio tikslumas**
*Algortimo **C4.5** modeliai*				
C45-1	*AIDS*, *PROTEIN*, *FINGER*, *POWER-N*	58620	15	17.6%
C45-2	*AIDS*, *MUTA*, *PROTEIN*, *GREC*, *LETTER*, *FINGER*	26853	15	26.52%
C45-3	*AIDS*, *MUTA*, *PROTEIN*, *GREC*, *LETTER*, *FINGER*	26853	15	20.7%
C45-4	*RANDOM-N*, *POWER-N*, *CLUSTER-N*	15925	15	11.78%
C45-5	*MUTA*, *PROTEIN*, *FINGER*, *RANDOM-N*, *POWER-N*	18970	5	36.91%
*Algoritmo **RepTree** modeliai*				
RepTree-1	*AIDS*, *FINGER*, *PROTEIN*, *POWER-N*	58620	15	17.63%
RepTree-2	*RANDOM-N*, *POWER-N*, *CLUSTER-N*	15925	15	11.87%
RepTree-3	*MUTA*, *PROTEIN*, *FINGER*, *RANDOM-N*, *POWER-N*	18970	5	36.49%

5 lentelė. Algoritmų *C4.5* ir *RepTree* sukonstruotų klasifikavimo modelių lentelė. Grafų rinkiniai *RANDOM-N*, *POWER-N*, *CLUSTER-N* simbolizuoja įvairių dydžių ir parametrų generuotus grafus, atitinkamai: atsitiktiniai, laipsninio dėsnio ir klasterizuoti grafų rinkiniai.

Šio darbo metu pastebėta, kad pasirinkimų medžiai, kurie naudojo įrašų atributus r_Δ ir d_Δ prastai pasirodę GRA skaičiavime atsižvelgiant į vykdymo laika. Taip nutinka, dėl to, kad norint apskaičiuoti grafo spindulį ar skersmenį, reikia rasti visų grafo viršūnių ekscentricitetus $e(v),\ v \in V$, kitaip sakant reikia išspręsti visų porų trumpiausių kelių uždavinį. Šiame darbe taikytas klasikinis Floido-Varšalo algoritmas visų porų trumpiausių kelių paieškai atlikti [Hou10]. Šio algoritmo vykdymo laiko sudėtingumas lygus $O(n^3)$, kas paaiškina GRA skaičiavimo suletėjimą minėtuose modeliuose. Kadangi GRA aplikacijose ypatingai svarbus algoritmų vykdymo laikas, šiame darbe pristatomuose pasirinkimų medžiuose nenaudojami atributai r_Δ ir d_Δ. Verta



1 pav. Algoritmo *C4.5* sukonstruoti pasirinkimų medžiai. Iliustracijoje naudojami GRA algoritmų trumpiniai: *Beam(BC*)* $\rightarrow$ *BP-Beam(Branch-Const)*, *Refine(BC*)* $\rightarrow$ *Refine(Branch-Const)*, *BC** $\rightarrow$ *Branch-Const*.



2 pav. Algoritmo *RepTree* sukonstruoti pasirinkimų medžiai. Iliustracijoje naudojami GRA algoritmų trumpiniai: *BC** $\rightarrow$ *Branch-Const*, *BF** $\rightarrow$ *Branch-Fast*.

paminėti, kad atributas j taip pat nevyrauja nei viename iš pristatytų modelių. Taip gali nutikti, kai atributas nėra informatyvus, kitaip pasakius nėra jokios koreliacijos tarp šio atributo ir įrašų klasės. Konstruojant pasirinkimų medį *C45-5*, buvo naudotos tik penki geriausiai pasirodę GRA algoritmai realaus pasaulio grafų eksperimentuose: $Y = \{$ *Star-5*, *Branch*, *Branch-Fast*, *BP*, *Refine(BP)* $\}$.

5. Rezultatai

Šiame poskyryje pateikiami *IPFP* ir su juo susijusių algoritmų empirinio tyrimo rezultatai ir analizė. Visi eksperimentai buvo atliekami vientisoje aplinkoje - programinio kodo karkase *GEDLIB* [BBG+19]. Tai kodo bazė skirta darbui su grafais ir grafo redagavimo atstumo paieškos algoritmais. Visi algoritmai yra įgyvendinti naudojantis C++ programavimo kalba ir naudojantis tomis pačiomis bazinėmis duomenų struktūromis. Algoritmų įgyvendinimas naudojantis tomis pačiomis duomenų struktūromis ir programavimo paradigma leidžia tiksliau palyginti algoritmų vykdymo laiko skirtumus. Kitaip sakant, galima būtų optimizuoti pavienių algoritmų implementaciją, tačiau tame mažai prasmės, jeigu tikslas lyginti pačius algoritmus, o ne jų implementacijas.

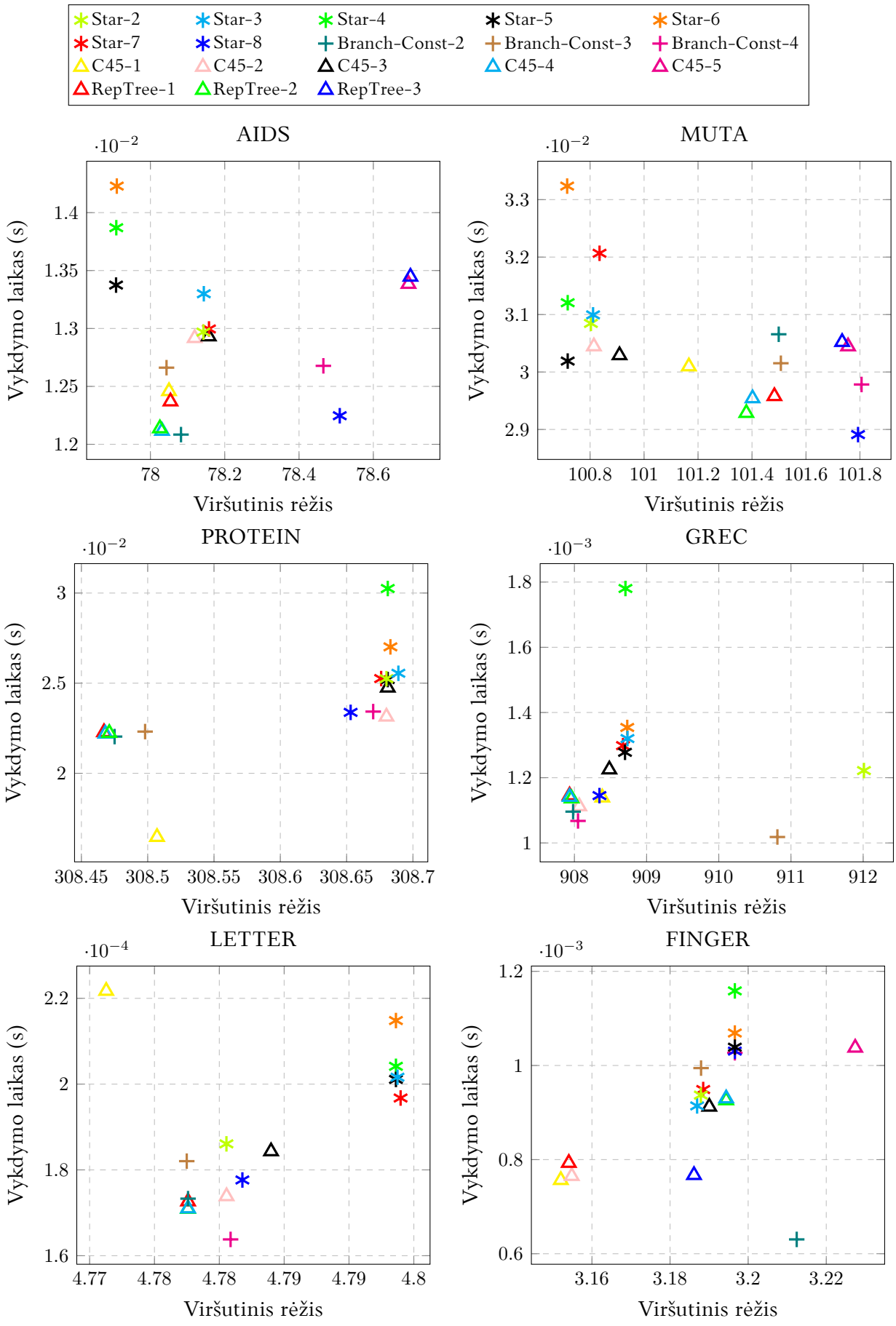
5.1. Eksperimentų konfigūracija

Pagrindinis testų subjektas yra euristinis LP-GRA klasės algoritmas *IPFP*. Šiam algoritmui svarbūs du parametrai: maksimalus leistinas iteracijų skaičius I ir sprendinio konvergavimo slenkstis ϵ. Kaip pasiūlyta originaliame *IPFP* darbe [BGB16], maksimalus iteracijų skaičius naudotas šiuose testuose yra $I = 100$ ir konvergavimo slenkstis $\epsilon = 10^{-3}$. Kadangi lokalios paieškos algoritmų rezultatai stipriai priklauso nuo pradinio sprendinio, dažnai tokie algoritmai vykdomi N kartų pradedant nuo skirtingų pradinių sprendinių. Atlikus visas ar dalį paieškų, gražinamas geriausias rezultatas. Tokia lokalios paieškos taktika gerai veikia, kadangi ją labai paprasta vykdyti paraleliai - tiesiog kiekvienas lokalios paieškos skaičiavimas paleidžiamas atskiroje gijoje. Šiame darbe, algoritmas *IPFP* buvo vykdomas tik viena kartą duotiems grafams - tai yra, nebuvo taikoma N pradinių sprendinių taktika *MultiStart* [BBB18]. Eksperimentuose duotam grafų rinkiniui R, grafų redagavimo atstumo paieškos algoritmai buvo leidžiami kiekvienai grafų $G \in R$ ir $H \in R$ kombinacijai. Daugumoje šio darbo eksperimentų buvo naudojami grafų rinkiniai sudaryti iš $|R| = 100$ skirtingų grafų. Tai reiškia, kad vienai algoritmų kombinacijai rinkinyje R yra atliekama $|R| \cdot |R| = 100 \cdot 100 = 10000$ atskirų skaičiavimų. Visi skaičiavimų rezultatai ir vykdymo laikas yra sumuojami ir galiausiai randami viršutinio rėžio ir vykdymo laiko vidurkiai, kurie toliau vaizduojami eksperimentų iliustracijose.

Visi eksperimentai buvo vykdomi kompiuteryje su Intel Xeon W-2145 procesoriumi ir 32 GB operatyviosios atminties. Šio darbo eksperimentus galima replikuoti programinio kodo repozitorijoje "GitHub" adresu: `https://github.com/karolis-zukauskas/gedlib`. Kadangi šioje repozitorijoje saugomi ir kitų darbų testai, norint replikuoti tik šio darbo eksperimentus reiktų sukompiliuoti ir įvykdyti "*kazu*" projekto programas.

5.2. Realaus pasaulio grafų eksperimentai

Pirma, šiame darbe yra apžvelgiamos visos pristatytos TSPKT-GRA klasės algoritmų modifikacijos ir sukonstruoti pasirinkimų medžių modeliai. Šiame eksperimente, modifikacijos yra pritaikomos pradiniam GRA sprendiniui surasti, kuris toliau perduodamas lokalios paieškos algoritmui *IPFP*. Visi algoritmai buvo testuojami su realaus pasaulio grafų rinkiniais, apibrėžtais lentelėje 2. Kadangi šiame darbe yra nagrinėjamas didelis kiekis įvairių euristikų grafų reda-



3 pav. *IPFP* algoritmo eksperimentai naudojant GRA algoritmų modifikacijas bei pasirinkimų medžius pradiniam lokalios paieškos sprendiniui parinkti. Eksperimentai realaus pasaulio grafų rinkiniuose: *AIDS*, *MUTA*, *PROTEIN*, *GREC*, *LETTER*, *FINGER*. Diagramų X ašis atitinka vidutinį GRA viršutinį rėžį, Y ašis - vidutinį algoritmo vykdymo laiką.

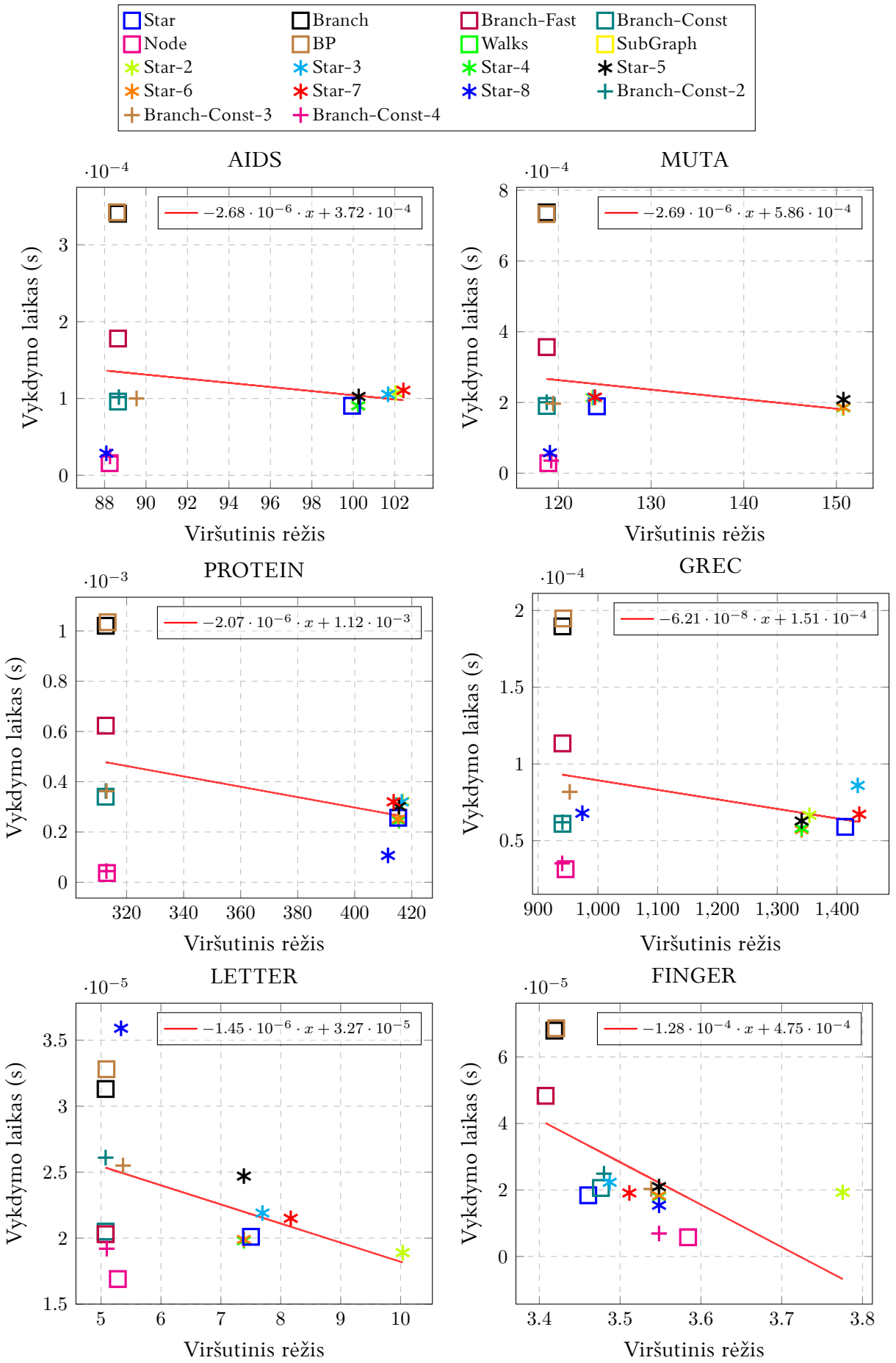
gavimo atstumui skaičiuoti, šioje eksperimentų dalyje yra apžvelgiamos tiktais GRA algoritmų modifikacijos. Tolimesniuose eksperimentuose ir jų iliustracijose, bus įtraukiamos tiktais geriausius rezultatus gražinančios euristikos, siekiant neperpildyti diagramų su persiengiančiais taškais.

Iliustracijoje 3 trikampio simboliu žymimi pasirinkimų medžiais paremti algoritmai, žvaigždute žymimos septynios algoritmo *Star* modifikacijos, o likusios trys algoritmo *Branch-Const* modifikacijos yra žymimos pliuso simboliu. Grafų rinkiniuose *AIDS* ir *MUTA* geriausiai pasirodė algoritmo *Star* modifikacijos. Šiuose rinkiniuose, trys modifikacijos gražino tiksliausius GRA rėžius tačiau iš jų *Star-5* taip pat turėjo geriausią vykdymo laiką. Rinkiniuose *PROTEIN* ir *GREC* geriausius rezultatus gražino pasirinkimo medžiais paremti algoritmai. Čia keli pasirinkimų modeliai gražino panašius rėžius ir turėjo labai panašius vykdymo laikus, tačiau tiksliausias modelis yra *RepTree-1*. Verta pastebėti, kad šiuose rinkiniuose gerai pasirodė ir TSPKT-GRA klasės algoritmo *Branch-Const-2* modifikacija, kuri paprastai gražino šiek tiek prastesnius rėžius nei pasirinkimų medžiai, bet turėjo šiek tiek geresnį vykdymo laiką. Realaus pasaulio rinkiniuose su nedideliais grafais *LETTER* ir *FINGER* geriausiai pasirodė pasirinkimų medžiai *C45-1*, *RepTree-1* ir modifikacija *Branch-Const-2*. Šiuose rinkiniuose algoritmo *Star* modifikacijos nepranoko kitų šiame darbe pristatytų, modifikacijų.

Toliau, šiame darbe yra apžvelgiami eksperimentai, kuriuose buvo testuojami TSPKT-GRA klasės algoritmai skaičiuojant viršutinį grafų redagavimo atstumo rėžį. Tai vienintelis eksperimentas šiame darbe, kuris nevykdo algoritmo *IPFP*, tačiau svarbus eksperimentas, siekiant nustatyti, kokie yra pavienių algoritmų sąryšiai ir kaip jų rezultatai keičiasi *IPFP* kontekste - kitaip sakant, juos pritaikius pradiniam sprendiniui parinkti. Iliustracijoje 4, kvadrato simboliu žymimi nemodifikuoti TSPKT-GRA klasės algoritmai, žvaigždute žymimi modifikuoti *Star* algoritmai ir *Branch-Const* modifikacijos žymimos pliuso simboliu. Iliustracijos legendoje paminėti algoritmai *Walks* ir *SubGraph* tačiau diagramose šie algoritmai nepavaizduoti dėl to, kad jų vykdymo laikas vidutiniškai keliomis eilėmis prastesnis už visus likusius algoritmus, o gražinami viršutiniai rėžiai nei viename iš grafų rinkinių nebuvo tarp geriausių algoritmų. Siekiant aiškiau pademonstruoti likusių algoritmų pasiskirstymą grafikuose, *Walks* ir *SubGraph* nėra įtraukti šio eksperimento į diagramas.

Iš šio eksperimento galima pastebėti, kad algoritmas *Star* ir beveik visos jo modifikacijos negražino tikslių GRA viršutinio rėžio aproksimacijų lyginant su kitomis TSPKT-GRA alternatyvomis. Vienintelė modifikacija *Star-8* dalyje grafų rinkinių gražino ganėtinai tikslius rezultatus. Modifikacija *Star-8* yra algoritmų *Star* ir *Node* kombinacija. Kadangi algoritmas *Node* beveik visuose grafų rinkiniuose gražino tiksliausius GRA rėžius, tai paaiškina modifikacijos *Star-8* rezultatus. Deja, bendru atveju *Node* pranoksta modifikaciją *Star-8*. Nors *Star* algoritmų vykdymo laikas yra greitesnis už daugumą likusių algoritmų, *IPFP* algoritmo kontekste tai neturi beveik jokios įtakos, kaip pastebima tolimesniame poskyryje.

Beveik visuose grafų rinkiniuose *Branch* pobūdžio algoritmai ir algoritmas *Node* gražino tiksliausius GRA rėžius. Visuose grafų rinkiniuose, algoritmų išsidėstymas laiko atžvilgiu vienodas - greičiausias algoritmas *Node*, lėčiausias *BP*. Čia verta atkreipti dėmesį į modifikaciją *Branch-Const-4* - visuose grafų rinkiniuose ši modifikacija prilygo algoritmui *Node* vykdymo laiku ir



4 pav. TSPKT-GRA algoritmų klasės eksperimentai realaus pasaulio grafų rinkiniuose: *AIDS*, *MUTA*, *PROTEIN*, *GREC*, *LETTER*, *FINGER*. Diagramų X ašis atitinka vidutinį GRA viršutinį rėžį, Y ašis - vidutinį algoritmo vykdymo laiką. Raudona linija žymi tiesinę regresiją.

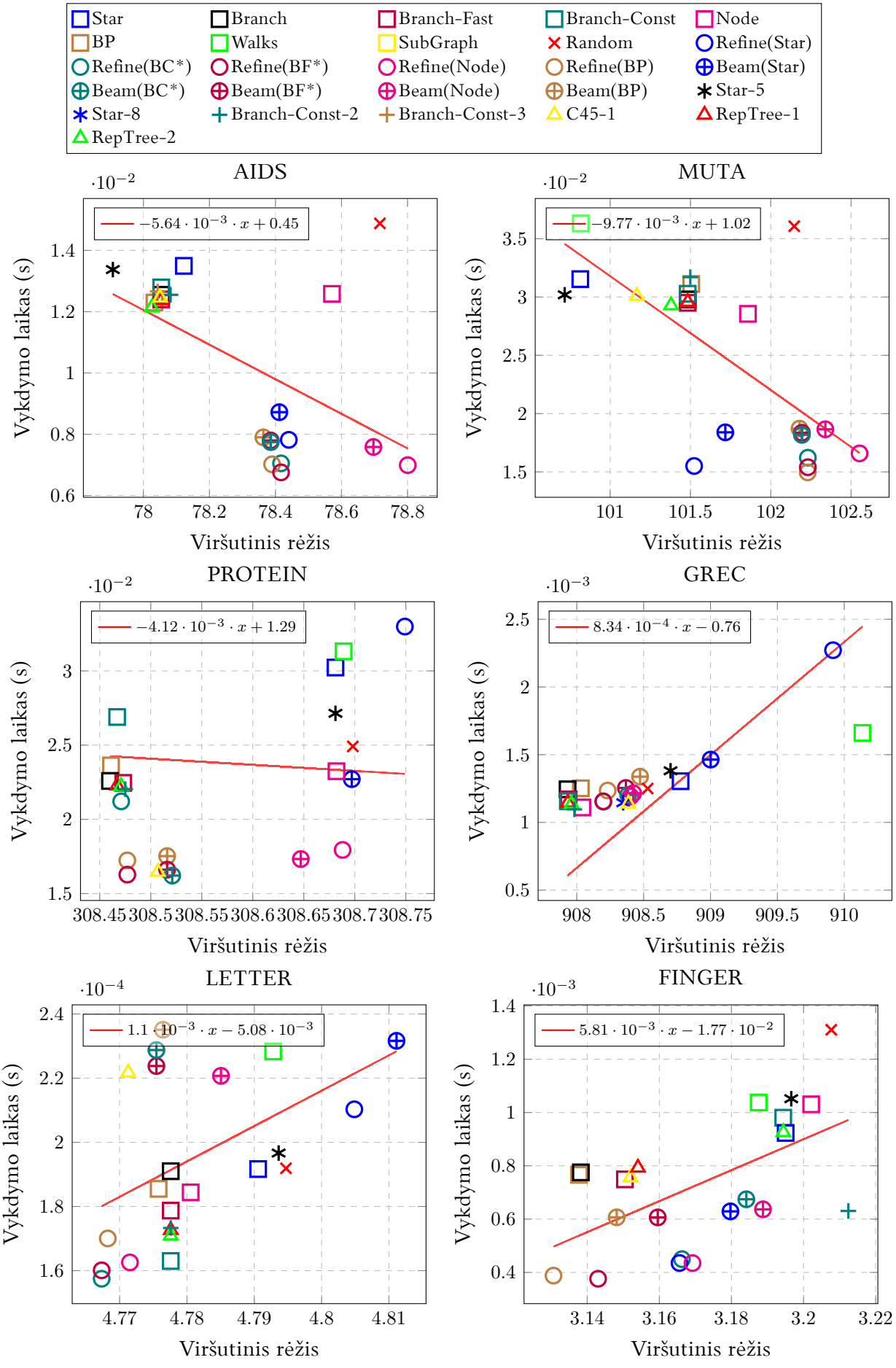
gražino identiškus arba tikslesnius grafo redagavimo atstumo rėžius. Ši algoritmo modifikacija yra geras kompromisas tarp algoritmo išgaunamų rėžių tikslumo ir vykdymo laiko.

Diagramose raudona linija žymi tiesinę regresiją. Visose diagramose regresijos linija yra tos pačios krypties, kuri sufleruoja, kad kuo didesnis algoritmo vykdymo laikas, tuo tikslesnis GRA viršutinis rėžis. Tačiau penkiuose iš šešių diagramų regresijos linija nėra labai stati, todėl vykdymo laiko ir viršutinio rėžio sąryšis nėra labai stiprus. Tai galima pastebėti pažvelgus į tiksliausius rėžius gražinančius algoritmus - rėžių skirtumai labai nedideli tačiau vykdymo laiko skirtumai dideli.

Sekančiame eksperimente buvo testuojamas *IPFP* algoritmas naudojant įvarius euristinius algoritmus pradiniam sprendiniui parinkti. Testai buvo atliekami su realaus pasaulio grafų rinkiniais, apibrėžtais lentelėje 2. Iliustracijoje 5 skirtingi simboliai žymi skirtingus algoritmus, kurie buvo naudoti inicializuoti pradiniam *IPFP* sprendiniui. Kvadrato simboliu žymimi nemodifikuoti TSPKT-GRA klasės algoritmai, žvaigždute žymimi modifikuoti *Star* algoritmai, pliuso simboliu žymimi modifikuoti *Branch-Const* algoritmai, apskritimais žymimi lokalios paieškos algoritmų *Refine* ir *Beam* variantai, trikampio simboliais žymimi pasirinkimų medžiais paremti algoritmai ir galiausiai kryželio simbolis atitinka atsitiktinai parinkta pradini sprendinį algoritmui *IPFP*. Iliustracijos legendoje paminėtas algoritmas *SubGraph* tačiau diagramose šis algoritmas neįtrauktas, kadangi jo vykdymo laikas ir gaunami rėžiai smarkiai nutolę nuo likusių algoritmų. Taip pat, ne visos šiame darbe pristatytos GRA modifikacijos yra įtrauktos į šio eksperimento diagramas. Atsižvelgiant į iliustraciją 3, buvo įtrauktos tik geriausius rezultatus gražinančios algoritmų modifikacijos.

Grafų rinkiniuose *AIDS* ir *MUTA*, kelios algoritmo *Star* modifikacijos gražino tiksliausius rėžius, iš kurių *Star-5* abejuose rinkiniuose buvo greičiausia. Rinkinyje *AIDS*, *Star-5* gražinamas rėžis yra sąlyginai toli nutolęs nuo konkuruojančių algoritmų *BP* ir alternatyvių *Branch* algoritmų, kurie likusiuose grafų rinkiniuose vidutiniškai gražino tiksliausius grafo redagavimo atstumo rėžius. Greitas lokalios paieškos algoritmas *Refine* veikia sąlyginai gerai siekiant paspartinti *IPFP* algoritmo vykdymo laiką. Kadangi *Refine* yra paremtas lokalia paieška, pradinio sprendinio parinkimas šiam algoritmui taip pat svarbus. Galima pastebėti, kad TSPKT-GRA algoritmai ir *Refine* kombinacijos su jais seka ta pačia tendencija. Pavyzdžiui jeigu algoritmas *Star* gražina tiksliausius rėžius tarp TSPKT-GRA klasės algoritmų, tai *Refine(Star)* gražina tiksliausią rėžį tarp kitų *Refine* kombinacijų - šį dėsningumą galima pastebėti visuose šešiuose grafų rinkiniuose. Kitas, lokalia paieška paremtas, algoritmas *Beam* nei viename grafų rinkinyje nepranoko algoritmo *Refine* kombinacijų nei vykdymo laiku nei išgaunamų GRA rėžių tikslumu. Didesniuose grafų rinkiniuose, *Refine* algoritmo kombinacijos negražino tiksliausių rėžių, tačiau mažo dydžio grafo rinkiniuose *LETTER* ir *FINGER* šios kombinacijos gražino tiksliausius rėžius ir taip pat aplenkė kitus algoritmus vykdymo laiku. Bendru atveju, visuose rinkiniuose greičiausius vykdymo laikus išgavo algoritmo *Refine* kombinacijos.

Čia galima įžvelgti įdomų pastebėjimą - natūraliai atrodytų, kad kuo tikslesnis pradinis grafų redagavimo atstumo sprendinys, tuo geresnis lokalios paieškos algoritmo, šio tyrimo atveju *IPFP*, rezultatas. Iš šių eksperimentų matome, kad idėja taikyti greitesnį, bet ne tokį tikslų, lokalios paieškos algoritmą, bandant parinkti pradinį sprendinį tikslesniam lokalios paieškos algoritmui



5 pav. *IPFP* algoritmo eksperimentai naudojant skirtingus algoritmus pradiniam lokalios paieškos sprendiniui parinkti. Eksperimentai realaus pasaulio grafų rinkiniuose. Diagramų X ašis atitinka vidutinį GRA viršutinį rėžį, Y ašis - vidutinį algoritmo vykdymo laiką. Raudona linija žymi tiesinę regresiją.

neveikia. Galimai taip nutinka dėl to, kad *Refine* algoritmas randa lokalų grafų redagavimo atstumo minimumą ir tikslesnis *IPFP* algoritmas nebesugeba ištrūkti iš duoto pradinio paieškos taško. Tuo tarpu pradedant nuo prastesnio sprendinio, tikslesnis lokalios paieškos algoritmas gali pasirinkti geresnę paieškos kryptį, kuri galimai nuves iki geresnio lokalaus minimumo - tikslesnio grafų redagavimo atstumo viršutinio rėžio.

Kitos šiame darbe pristatytos GRA modifikacijos ir pasirinkimų medžiai, atmetus *Star-5*, nepranoko nemodifikuotų algoritmų. Rinkiniuose *PROTEIN* ir *GREC* pasirinkimų medžiai *RepTree-1* bei *RepTree-2* gražino labai panašius rėžius, kaip algoritmo *Branch* variacijos, tiktais su šiek tiek geresniu vykdymo laiku. Čia nieko keisto, kadangi dauguma šių pasirinkimų medžių lapų atitinka kurią nors *Branch* variaciją.

Algoritmas	**AIDS**	**MUTA**	**PROTEIN**	**GREC**	**LETTER**	**FINGER**
Refine (BP)	6.50	5.53	6.42	2.67	2.42	3.65
Refine (Branch-Fast)	6.51	5.50	6.34	2.62	2.37	3.86
Refine (Branch-Const)	6.51	5.50	6.33	2.64	2.37	4.75
Refine (Node)	7.76	6.38	7.15	2.97	2.51	5.12
Refine (Star)	6.42	4.85	7.61	2.96	2.67	4.73
Beam (BP)	8.38	8.04	7.88	3.07	2.56	5.75
Beam (Branch-Fast)	8.34	8.10	7.61	3.00	2.60	5.92
Beam (Branch-Const)	8.34	8.10	7.58	3.01	2.61	7.41
Beam (Node)	9.08	8.35	8.08	3.29	3.01	8.01
Beam (Star)	9.73	8.00	10.79	5.51	4.16	7.09
BP	14.08	13.68	10.67	5.91	4.69	10.00
Branch	14.06	13.62	10.41	5.78	4.99	10.12
Branch-Fast	14.06	13.62	10.46	5.78	4.99	10.06
Branch-Const	14.06	13.62	10.42	5.77	4.99	14.40
Node	14.86	13.83	10.90	6.13	5.63	15.94
Random	16.42	16.04	12.19	7.43	7.12	18.10
Star	14.32	14.32	12.08	7.30	6.63	14.74
SubGraph	16.48	13.75	11.63	6.67	5.71	18.10
Walks	14.45	13.77	12.18	6.89	6.75	14.78

6 lentelė. Algoritmo *IPFP* vidutiniškai atliktų iteracijų skaičius realaus pasaulio grafų rinkinių eksperimente, su grafų rinkiniais, apibrėžtais lentelėje 2.

Šio eksperimento diagramose taip pat įtraukta tiesinės regresijos linija, pažymėta raudonai. Vidutinio dydžio grafo rinkiniuose *AIDS* ir *MUTA* regresijos linija yra ganėtinai stati ir neigiamos krypties, kas reikštų, kad algoritmai su ilgesniu vykdymo laiku išgauna tikslesnius rėžius. Likusiuose, mažesnio dydžio grafų rinkiniuose regresijos linija pakankamai stati ir teigiamos krypties, kas reikštų, kad kuo algoritmas greičiau baigia darbą tuo tikslesnį rėžį jis gražina. Iš pirmo žvilgsnio tai nebūtinai atrodo, kaip tikėtinas sąryšis, bet galimai tai paaiškinama atsižvelgus į kontekstą. Lokalios paieškos algoritmas *IPFP* iteratyviai vykdo paiešką iki tol, kol surandą lokalų minimumą arba pasiekia maksimalų leistina iteracijų skaičių. Šiame eksperimente, nei viena *IPFP* algoritmo kombinacija neviršijo maksimalaus leistino iteracijų skaičiaus ir vidutiniškai baigė darbą įvykdžius 10 paieškos iteracijų, kaip matoma lentelėje 6. Taip pat, duotas sąryšis pastebimas tik mažo dydžio grafų rinkiniuose, kas reiškia mažesnę paieškos erdvę - galimai tokioje erdvėje, lokalių

minimumų yra žymiai mažiau lyginant su didelių grafų paieškos erdve ir tikslesnis pradžios taškas greičiau priveda *IPFP* iki gero lokalaus minimumo.

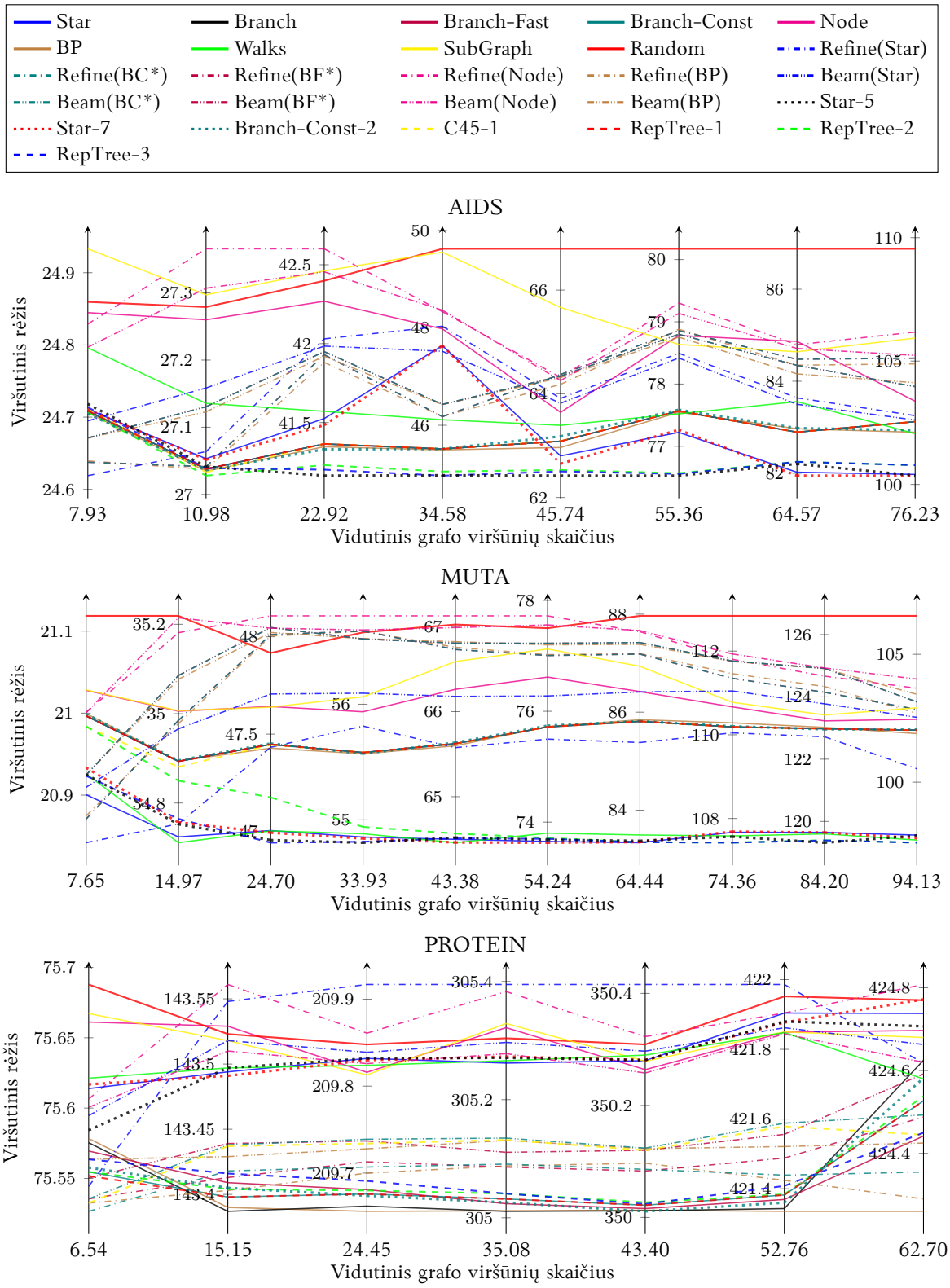
Lentelėje 6 pavaizduoti algoritmo *IPFP* vidutiniškai atliktų lokalios paieškos iteracijų skaičius realaus pasaulio grafų rinkinių eksperimente 5. Čia galima pastebėti, kad visos *Refine* algoritmų kombinacijos vidutiniškai atliko mažiausią *IPFP* lokalios paieškos iteracijų skaičių. Šis pastebėjimas dera su prieš tai daryta prielaidai, kad *Refine* algoritmas gražina pakankamai gerą pradinį sprendinį, lokalų grafų redagavimo atstumo minimumą, iš kurio *IPFP* neištrūksta ir greitai baigia savo darbą pasiekus sprendinio konvergavimo slenkstį ϵ. Testuojant algoritmą *SubGraph*, *IPFP* vidutiniškai atliko panašų kiekį iteracijų, kaip ir kiti TSPKT-GRA klasės algoritmai, tačiau šios kombinacijos vykdymo laikas vidutiniškai buvo penkis kartus lėtesnis. Tai reiškia, kad pradinio sprendinio paieška užtruko žymiai ilgiau, nei likusiųjų algoritmų.

5.3. Skaidytų realaus pasaulio grafų eksperimentai

Šiame eksperimente algoritmas *IPFP* buvo testuojamas skaldytų realaus pasaulio grafų rinkiniuose. Realaus pasaulio grafų rinkiniai *AIDS*, *MUTA* ir *PROTEIN* buvo išskaldyti į $N_i \;\; i \in \{1, \ldots, m\}$ grafų grupes, kur kiekviena grupė sudaryta iš $10 \leq |N_i| \leq 100$ atsitiktine tvarka parinktų grafų, taip kad kiekvienam grupės N_i grafui G būtų teisinga lygybė $\lfloor \frac{|V_G|}{10} \rfloor = i-1 \;\; G \in N_i$. Kitaip sakant, kiekvienoje grafų grupėje, grafų dydžiai patenka į ta pačia dešimtį: $1 \leq |V_G| \leq 10, \; G \in N_1$, $11 \leq |V_G| \leq 20, \; G \in N_2$ ir taip toliau.

Iliustracijoje 6 vaizduojamos paralelių koordinačių diagramos, kur kiekviena vertikali ašis žymi vidutinius grafų redagavimo atstumo viršutinius rėžius, gražintus algoritmo *IPFP* kombinacijų skirtingoms grafų grupėms N_i. Skaičius kiekvienos vertikalios ašies apačioje žymi grupės N_i grafų vidutinį viršūnių skaičių. Pilnomis linijomis žymimi nemodifikuoti TSPKT-GRA klasės algoritmai, taškuotomis linijomis žymimos šio darbo TSPKT-GRA klasės algoritmų modifikacijos, brūkšninės linijos žymi pasirinkimų medžiais paremtus algoritmus, brūkšniuotos-taškuotos linijos atitinka algoritmų *Refine* ir *Beam* variacijas ir galiausiai pilna, raudona linija žymi atsitiktinai parinkto pradinio *IPFP* sprendinio rezultatus.

Taip pat kaip eksperimente su neskaldytu grafų rinkiniu *AIDS*, šiame eksperimente matosi, kaip modifikacija *Star-5* gražina tiksliausius rėžius beveik visose grafų grupėse išskyrus pirmą grupe kur vidutinis grafų viršūnių skaičius yra mažas. Diagramoje su *MUTA* rinkinio grafų grupėmis, ši modifikacija taip pat vidutiniškai gražino tiksliausius rėžius vidutinio ir didelio dydžio grafams. Lyginant su neskaidytų grafų rinkinių eksperimentu, pasirinkimų medis *RepTree-1* gražino prastesnius rėžius nei kiti du algoritmo *RepTree* modeliai. Skaidytuose rinkiniuose *AIDS* bei *MUTA* pasirinkimų medis *RepTree-3* gražino panašius rėžius, kaip modifikacija *Star-5* - tai yra tiksliausius rėžius iš visų kitų algoritmų. Taip pat, kaip neskaidytų rinkinių eksperimente, algoritmas *Walks* parodė gerus rezultatus tik rinkinyje *MUTA*, tačiau nepranoko algoritmų *Star-5* ar *RepTree-1*. Grafų rinkinyje *PROTEIN*, tiksliausią grafų redagavimo atstumo viršutinį rėžį gražino algoritmas *BP*, nuo kurio toli neatsiliko *Branch*. Svarbu pastebėti, kad visuose rinkiniuose, mažų grafų grupėse, lokalios paieškos algoritmų *Refine* bei *Beam* variacijos gražino tiksliausius rė-



6 pav. *IPFP* algoritmo eksperimentai naudojant skirtingus algoritmus pradiniam lokalios paieškos sprendiniui parinkti skaldytuose realaus pasaulio grafų rinkiniuose: *AIDS*, *MUTA*, *PROTEIN*. Diagramų Y ašys atitinka vidutinį GRA viršutinį rėžį skirtingo vidutinio dydžio grafuose. Skaičius ašių apačioje atitinka vidutinį grafų viršūnių kiekį.

žius, tačiau augant grafų viršūnių kiekiui šie algoritmai yra pralenkiami kitų TSPKT-GRA klasės algoritmų.

5.4. Atsitiktinai generuotų grafų eksperimentai

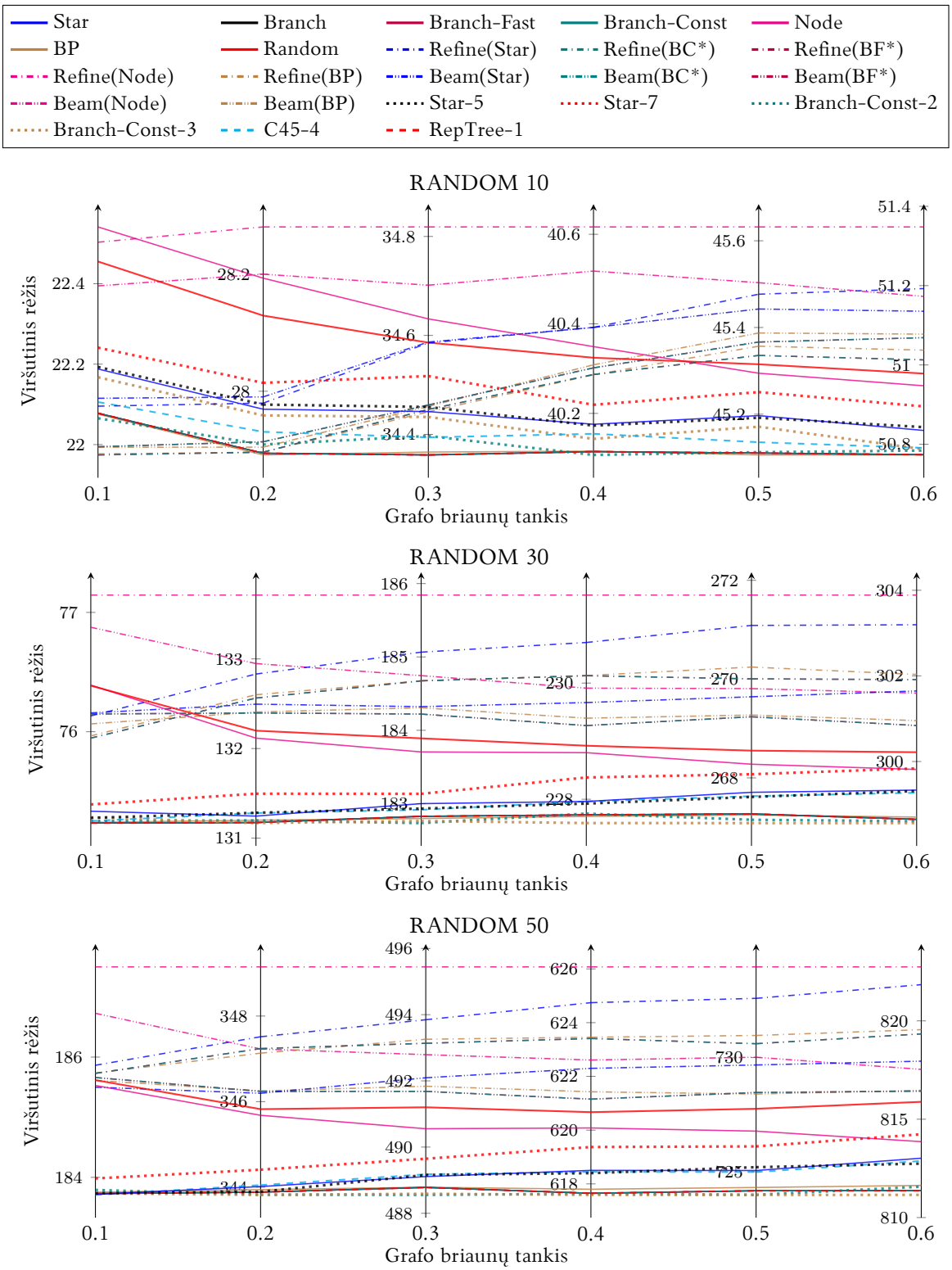
Šiame eksperimente algoritmas *IPFP* buvo testuojamas atsitiktinai generuotuose grafuose. Iš viso buvo generuoti trys skirtingų grafų dydžių rinkiniai. Kiekvienas rinkinys toliau skaidomas į pogrupius, kurie skiriasi grafų briaunų tankiais. Kiekvieną grafų pogrupį sudaro 100 atsitiktinai generuotų grafų. Iliustracijoje 7 vaizduojamos trys paralelių koordinačių diagramos skirtos skirtingo dydžio grafų rinkiniams. Diagramose *RANDOM N* skaičius *N* simbolizuoja grafų rinkinio vidutinį grafų viršūnių skaičių. Kiekviena vertikali diagramos ašis žymi vidutinius grafų redagavimo atstumo viršutinius rėžius, gražintus algoritmo *IPFP* kombinacijų skirtingo tankio grafams. Skaičius kiekvienos vertikalios ašies apačioje žymi grafų briaunų tankį $0 < t < 1$, kuris žymi procentinę dalį maksimalaus briaunų skaičiaus neorientuotame grafe G. Tikslus briaunų skaičius duotam grafui G ir briaunų tankiui t apskaičiuojamas taip: $|E_G| = \lfloor t \times \frac{|V_G| \times |V_G| - |V_G|}{2} \rfloor$. Pilnomis linijomis žymimi nemodifikuoti TSPKT-GRA klasės algoritmai, taškuotomis linijomis žymimos šio darbo TSPKT-GRA klasės algoritmų modifikacijos, brūkšninės linijos žymi pasirinkimų medžiais paremtus algoritmus, brūkšniuotos-taškuotos linijos atitinka algoritmų *Refine* ir *Beam* variacijas ir galiausiai pilna, raudona linija žymi atsitiktinai parinkto pradinio *IPFP* sprendinio rezultatus.

Iš šio eksperimento galima pastebėti, kad algoritmo *Refine* variacijos gražino tiksliausią viršutinį rėžį tik mažiausio dydžio grafų rinkinyje *RANDOM 10* ir esant nedidesniam, kaip 20% briaunų tankiui, kas išsiverčia į apytiksliai $|E_G| \approx 9$ briaunas per grafą. Visose kitose grafų dydžio ir tankio kombinacijos algoritmai *Refine* negražino tikslių rezultatų - tokiuose grafuose geresnius rezultatus išgauna atsitiktinai parinktas pradinis sprendinys.

Panašiai, kaip ir kai kuriems realaus pasaulio grafų rinkiniams, algoritmo *Branch* variacijos gražino tokius pačius rėžius nepriklausomai nuo grafo dydžio ar jo tankio visiems atsitiktinai generuotiems grafų rinkiniams. Pasirinkimų medžiais paremto algoritmo *RepTree-1* rezultatai sutampa su *Branch* algoritmais. Grafų rinkinyje *RANDOM 10* šie algoritmai, kartu su algoritmu *BP* vidutiniškai gražino tiksliausią grafų redagavimo atstumo viršutinį rėžį. Tuo tarpu didesnio grafo viršūnių skaičiaus rinkiniuose *RANDOM 30* ir *RANDOM 50* algoritmo *Branch-Const* modifikacijos gražino pačius tiksliausius GRA rėžius.

5.5. Laipsninio dėsnio grafų eksperimentai

Šiame eksperimente algoritmas *IPFP* buvo testuojamas su grafais, kurie atitinka laipsninį dėsnį (angl. power law). Šiame darbe laipsninio dėsnio grafai buvo generuojami modeliu, kuris iteratyviai konstruoja grafą, kiekvienos iteracijos metu prijungiant N naujų briaunų. Iliustracijose, *POWER-N* atitinka grafų rinkinius, kur N simbolizuoja generavimo metu prijungiamas briaunas. Akivaizdu, kad kuo didesnis N, tuo tankesnis grafas yra sugeneruojamas - lentelėje 7 pavaizduoti vidutiniai briaunų skaičiai įvairaus viršūnių kiekio grafuose. GRA algoritmų rezultatai lyginami pasinaudojant paralelių koordinačių diagramomis, kurios pavaizduotos iliustracijoje 8. Skaičius



7 pav. *IPFP* algoritmo eksperimentai naudojant skirtingus algoritmus pradiniam lokalios paieškos sprendiniui parinkti atsitiktinai generuotuose grafuose. Diagramų Y ašys atitinka vidutinį GRA viršutinį rėžį skirtingo vidutinio dydžio grafuose.

kiekvienos vertikalios ašies apačioje žymi grafų vidutinį viršūnių skaičių, šiame eksperimente buvo atlikti testai su šešiais skirtingais grafų viršūnių kiekiais $|V| \in \{10, 20, \ldots, 60\}$. Iš viso buvo generuojama po 50 laipsninio dėsnio grafų duotai N ir $|V|$ kombinacijai - grafų redagavimo atstumas buvo skaičiuojamas visoms galimoms grafų poroms, kas atitinka 2500 atskirų GRA skaičiavimų. Pilnomis linijomis žymimi nemodifikuoti TSPKT-GRA klasės algoritmai, taškuotomis linijomis žymimos šio darbo TSPKT-GRA klasės algoritmų modifikacijos, brūkšninės linijos žymi pasirinkimų medžiais paremtus algoritmus, brūkšniuotos-taškuotos linijos atitinka algoritmų *Refine* ir *Beam* variacijas ir galiausiai pilna, raudona linija žymi atsitiktinai parinkto pradinio *IPFP* sprendinio rezultatus.

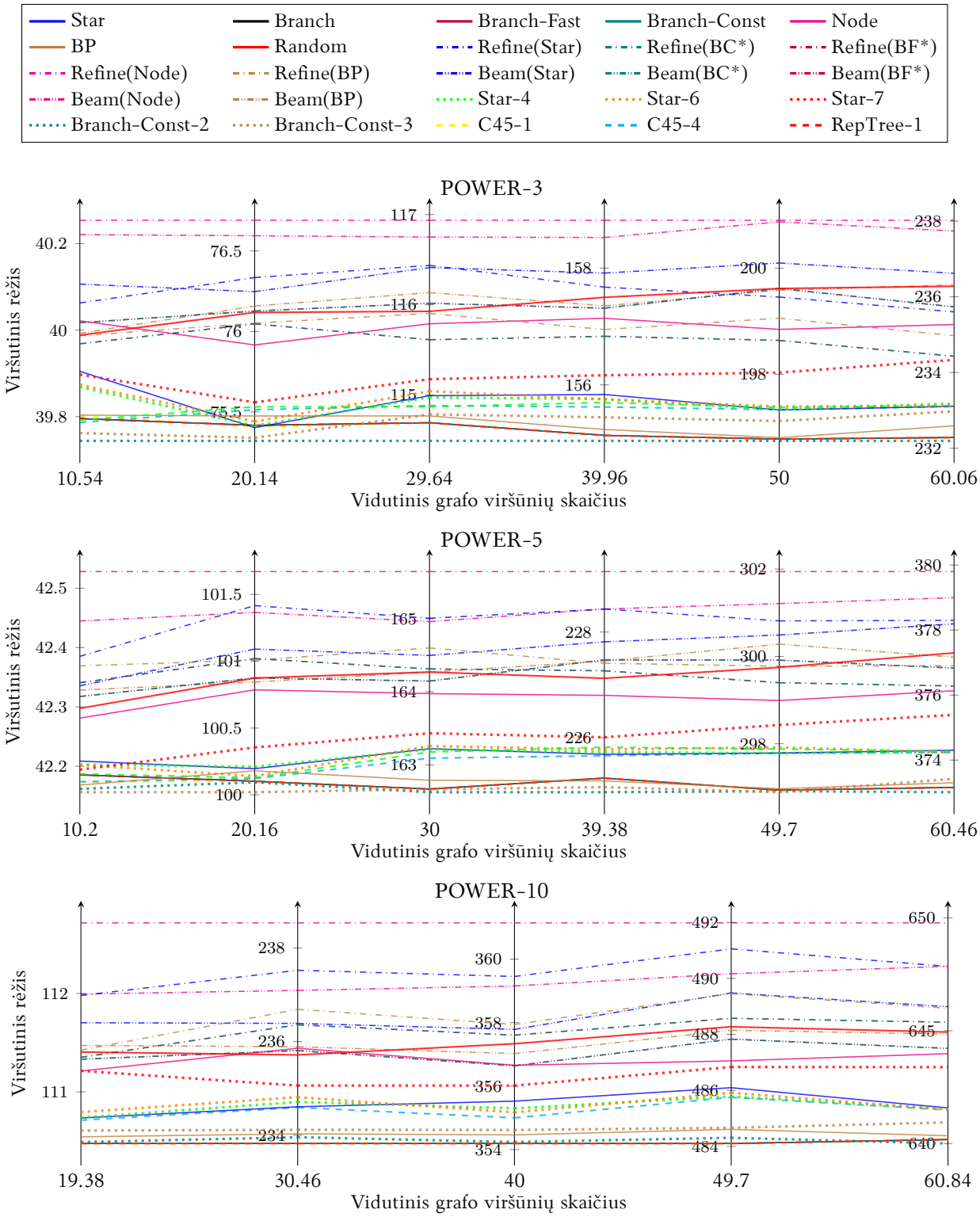
Rinkinys	**Vidutinis viršūnių sk.**					
	10	20	30	40	50	60
POWER-3	21.62	50.42	78.92	109.88	140	170.18
POWER-5	23	72.8	122	168.9	220.5	274.3
POWER-10	-	85.8	196.6	292	389	500.4

7 lentelė. Laipsninio dėsnio grafų rinkinių grupių vidutiniai briaunų kiekiai.

Iš šio eksperimento galime pastebėti, kad algoritmų *Refine* ir *Beam* variacijos niekada negražino tiksliausių GRA rėžių ir dažnu atveju gražino prastesnius rezultatus negu atsitiktinai parinktas pradinis sprendinys. Daugumoje atveju, algoritmas *Beam* pranoko algoritmą *Refine*. Atkreipus dėmesį į geriausiai pasirodžiusius algoritmus, pastebime įvairias algoritmų *Branch* variacijas, iš kurių šiame darbe pristatyta *Branch-Const-2* bendru atveju gražino tiksliausius GRA rėžius. Kaip ir kituose eksperimentuose, nemodifikuotos algoritmo *Branch* versijos gražino beveik identiškus rezultatus ir diagramose šiuos algoritmus reprezentuojančios linijos persidengia. Pasirinkimų medžiais paremtas algoritmas *RepTree-1* taip pat gražino gerus rezultatus, kurie identiški nemodifikuotiems *Branch* algoritmams - čia nieko keisto, kadangi, kaip matoma pasirinkimų medžių modelių vizualizacijoje 2, beveik visi pasirinkimų medžio *RepTree-1* lapai parenka vieną iš *Branch* algoritmų.

5.6. Klasterizuotų grafų eksperimentai

Šiame eksperimente algoritmas *IPFP* buvo testuojamas su kalsterizuotais grafais, tai yra grafais, kurie turi c viršūnių klasterių. Grafo viršūnės priklausančios tam pačiam kalsteriui yra tankiai sujungtos briaunomis, o briaunos tarp viršūnių esančių skirtinguose klasteriuose yra labai retos. Konstruojant klasterizuotus grafus, apart viršūnių kiekio, reikia pasirinkti klasterių skaičių c bei dvi tikimybes P_c ir P_i, tai yra tikimybė briaunos egzistavimui tarp viršūnių tame pačiame klasteryje ir tikimybės briaunos egzistavimui tarp viršūnių iš skirtingų klasterių. Šiame eksperimente visiems klasterizuotiems grafams generuoti buvo naudojamos reikšmės: $P_c = 0.8$ ir $P_i = 0.02$. Reikšmė N grafų rinkiniuose *CLUSTER-N* atitinka klasterių grafuose skaičių c. Šiame darbe atliekami eksperimentai su trimis klasterizuotų grafų rinkiniais, kurie toliau skaidomi į šešias grafų grupes su skirtingais viršūnių kiekiais. Grafų viršūnių skaičius $|V|$ kiekvienoje grupėje yra paskaičiuojamas naudojantis formule $|V| = X + \lfloor 5P + \frac{1}{2} \rfloor$, kur P yra atsitiktinė reikšmė įgyjanti



8 pav. Algoritmo *IPFP* eksperimentai naudojant skirtingus algoritmus pradiniam lokalios paieškos sprendiniui parinkti laipsninio dėsnio grafų rinkiniuose. Diagramų Y ašys atitinka vidutinį GRA viršutinį rėžį skirtingo vidutinio dydžio grafuose. Skaičius ašių apačioje atitinka vidutinį grafų viršūnių kiekį.

reikšmės $-1 \leq P \leq 1$ ir $X \in \{20, 30, \ldots, 70\}$. Kitaip pasakius viršūnių kiekis kiekvienoje grafų rinkinio grupėje maksimaliai skirsis per 10 viršūnių. Briaunų kiekio statistika klasterizuotų grafų rinkiniuose yra pavaizduota lentelėje 8. Grafų rinkinys *CLUSTER-2* yra ypatingai tankus, tai yra galimų briaunų skaičius kiekviename iš rinkinio grupių apytiksliai siekia pusę visų galimų briaunų skaičiaus duotam grafo dydžiui. Iš viso buvo generuojama po 50 grafų kiekvienai grafų rinkinio grupei - grafų redagavimo atstumas buvo skaičiuojamas visoms galimoms grafų poroms, kas atitinka 2500 atskirų GRA skaičiavimų duotai grafų grupei.

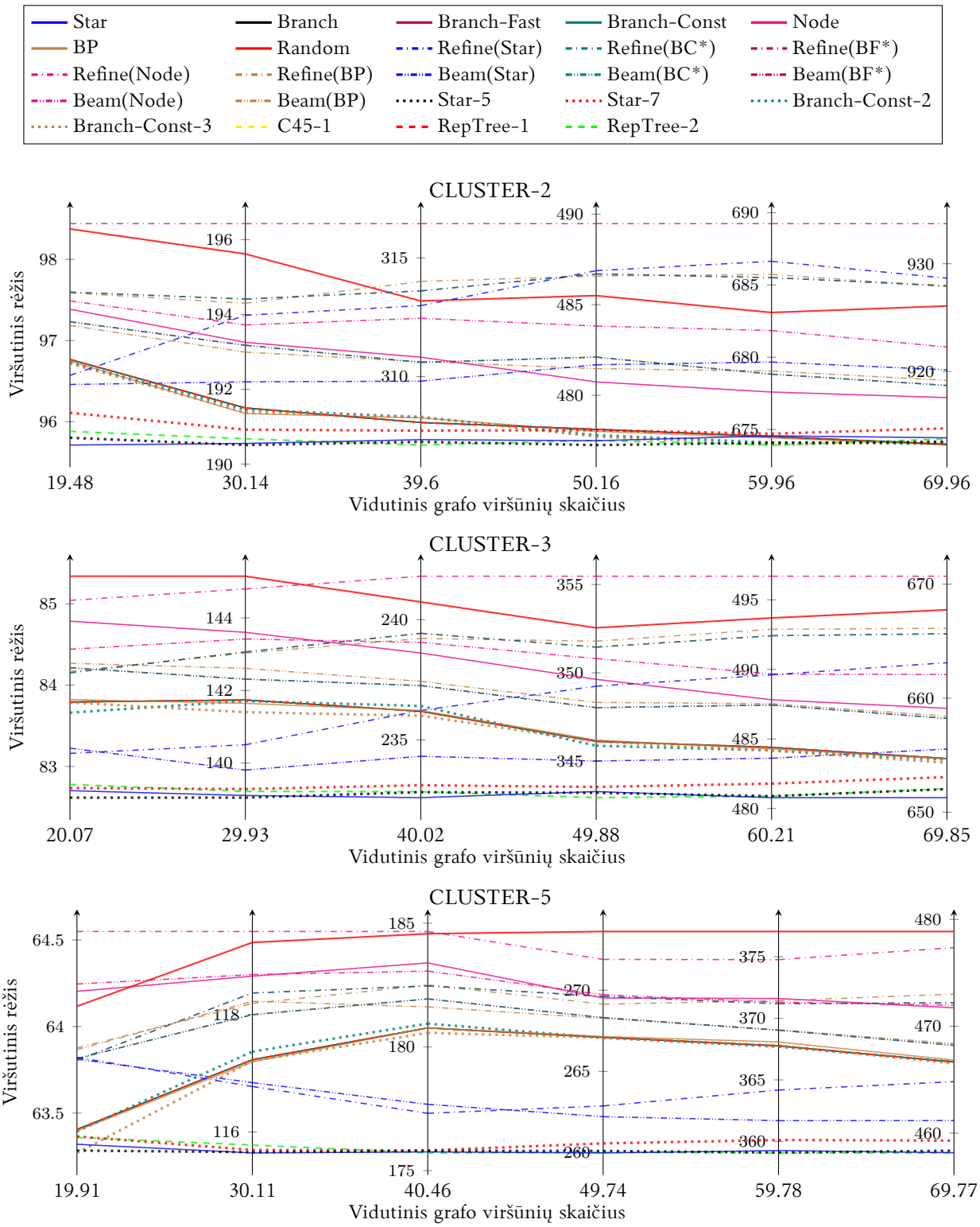
Rinkinys	**Vidutinis viršūnių sk.**					
	20	30	40	50	60	70
CLUSTER-2	87.36	213.38	371.74	604.6	861.34	1183.58
CLUSTER-3	62.18	135.26	252.02	402.1	579.86	797.22
CLUSTER-5	38.86	85.78	151.28	246.4	369.34	493.06

8 lentelė. Klasterizuotų grafų rinkinių grupių vidutiniai briaunų kiekiai.

Šio eksperimento rezultatai atvaizduojami iliustracijoje 9 naudojantis paralelių koordinačių diagramomis. Kiekviena vertikali diagramos ašis atitinka viena grafų rinkinio grupę, o skaičius ašies apačioje atitinką vidutinį grafų viršūnių kiekį toje grafų grupėje. Šiose diagramose pilnomis linijomis žymimi nemodifikuoti TSPKT-GRA klasės algoritmai, taškuotomis linijomis žymimos šio darbo TSPKT-GRA klasės algoritmų modifikacijos, brūkšninės linijos žymi pasirinkimų medžiais paremtus algoritmus, brūkšniuotos-taškuotos linijos atitinka algoritmų *Refine* ir *Beam* variacijas ir galiausiai pilna, raudona linija žymi atsitiktinai parinkto pradinio *IPFP* sprendinio rezultatus.

Šiame eksperimente, visai kitaip nei eksperimentuose su atsitiktinai generuotais ar laipsninio dėsnio grafais, tiksliausius grafo redagavimo atstumo rėžius gražino algoritmas *Star* ir jo modifikacija *Star-5*. Algoritmų variacijos *Branch* pasirodė ganėtinai prasčiau už *Star* algoritmus, tačiau augant grafo viršūnių skaičiui šių algoritmų išgaunamų rėžių tikslumas artėjo prie tiksliausiųjų, o rinkinyje *CLUSTER-2* susilygino su algoritmu *Star* ir jo modifikacijomis. Taip pat labai tikslius viršūnių rėžius išgavo pasirinkimų medis *RepTree-2*. Šis pasirinkimų medis iš viso turi penkis lapus, iš kurių trys parenka algoritmą *Branch-Const* ir likę du algoritmą *Star* bei vieną iš šio algoritmo modifikacijų *Star-5*. Čia įdomus pastebėjimas yra toks, kad paralelių koordinačių diagramose, algoritmą *Branch-Const* žyminti linija yra ganėtinai smarkiai nutolusi nuo *RepTree-2* žyminčios linijos, o tai reiškia, kad pasirinkimų medis dažniau parinkdavo *Star* algoritmus, kurie šiame eksperimente gražina geriausius rezultatus. Deja, laipsninio dėsnio grafų ar atsitiktinai generuotų grafų eksperimentuose, tas pats pasirinkimų medis, ar beveik identiškas *C45-4*, negražino tikslesnių rėžių už tuose eksperimentuose dominavusį *Branch-Const* algoritmą. Tai reiškia, kad šie pasirinkimų medžiai, negali nustatyti kada kuris iš euristinių algoritmų yra tinkamiausias grafų redagavimo atstumo skaičiavimui.

Čia galima palyginti laipsninio dėsnio ir klasterizuotų grafų eksperimentus. Pažvelgus į šių dviejų eksperimentų briaunų kiekių lenteles 7 ir 8 galima pastebėti, kad kai kurios generuotų grafų rinkinių grupės yra labai panašios viršūnių ir briaunų skaičiumi. Pavyzdžiui, rinkinių *POWER-3*



9 pav. Algoritmo *IPFP* eksperimentai naudojant skirtingus algoritmus pradiniam lokalios paieškos sprendiniui parinkti klasterizuotų grafų rinkiniuose. Diagramų Y ašys atitinka vidutinį GRA viršutinį rėžį skirtingo vidutinio dydžio grafuose. Skaičius ašių apačioje atitinka vidutinį grafų viršūnių kiekį.

ir *CLUSTER-5* grafų grupės, kur $|V| \approx 30$, arba rinkinių *POWER-5* ir *CLUSTER-5*, kur $|V| \approx 50$, beveik sutampa briaunų skaičiais. Palyginus šių rinkinių grupių GRA skaičiavimo rezultatus, matome, kad laipsninio dėsnio grafų rinkiniuose dominavo *Branch* algoritmo variacijos, tuo tarpu klasterizuotų grafų rinkiniuose dominavo algoritmas *Star* ir jo modifikacijos. Šie eksperimentai dar karta parodo, kad grafų redagavimo atstumo skaičiavimui svarbus ne tik grafo viršūnių ar briaunų skaičius, bet taip pat ir grafo struktūra.

Išvados

Šiame darbe pateikiami principiniai sprendimai grafų redagavimo atstumą aproksimuojančio lokalios paieškos algoritmo *IPFP* spartinimui, bei tikslinimui. Šio algoritmo vykdymo laikas bei išgaunamų grafų redagavimo atstumo rėžių tikslumas stipriai priklauso nuo pradinio sprendinio, kuris naudojamas lokaliai paieškai atlikti. Šiame darbe pateikiamas ir nagrinėjamas empirinis šio algoritmo ir skirtingų pradinio sprendinio parinkimo euristikų tyrimas, kuris parodo, kad gero pradinio sprendinio parinkimas nėra trivialiai atsakomas klausimas. Taip pat šiame darbe pristatomos modifikuotos, gerus rezultatus parodžiusios, TSPKT-GRA klasės algoritmų modifikacijos *Star-5*, *Branch-Const-2* ir *Branch-Const-4*.

Kadangi algoritmas *IPFP* yra keliomis eilėmis lėtesnis už daugumą TSPKT-GRA klasės algoritmų, norint rasti kompromisą tarp vykdymo laiko ir išgaunamų grafo redagavimo atstumo rėžių tikslumo, vertėtų naudoti šiame darbe pristatyta algoritmą *Branch-Const-4*, kuris lyginant su kitais TSPKT-GRA klasės algoritmais turėjo greičiausią vykdymo laiką, bei daugeliu atvejų buvo ir tiksliausias. Siekiant paspartinti algoritmo *IPFP* vykdymo laiką ar patikslinti jo išgaunamų grafų redagavimo atstumo rėžius reikia pasirinkti tinkamą, GRA euristinį algoritmą, kuris greitai surastų gerą pradinį GRA sprendinį. Nedidelio dydžio retiems grafams, tai yra grafams, kurių viršūnių skaičius yra apytiksliai $|V_G| \approx 10$ ir briaunų kiekis apytiksliai neviršija, $|E_G| \lessapprox 30$ reiktų naudoti algoritmus *Refine(BP)* ir *Refine(Branch-Fast)*, kadangi šių algoritmų parinkti pradiniai sprendiniai greičiausiai konverguoja į, tuo pačiu ir tiksliausius, grafų redagavimo atstumo rėžius. Vidutinio dydžio grafams, kurių viršūnių skaičius apytiksliai papuola į rėžius $15 \leq |V_G| \leq 30$ ir grafo briaunų skaičius nėra aukštas, tai yra apytiksliai neviršija 15% maksimalaus galimo briaunų skaičiaus, tiksliausius GRA rėžius išgauna šiame darbe pristatyto *Star-5* algoritmo sukonstruoti pradiniai sprendiniai. Galiausiai, skaičiuojant grafų redagavimo atstumą dideliuose ir tankiuose grafuose, tai yra grafuose, kur viršūnių skaičius yra $|V_G| \geq 30$ ir grafo briaunų skaičius viršija 15% maksimalaus galimo briaunų skaičiaus, geriausia naudoti algoritmus *BP*, *Branch-Fast* arba *Branch-Const-2* kadangi šie algoritmai geriausiai spartina ir tuo pačiu tikslina algoritmą *IPFP*.

Šiame darbe buvo nagrinėjama grafų struktūros įtaka grafų redagavimo atstumo skaičiavimui. Eksperimentuose pastebėta, kad skirtingos struktūros, bet panašaus viršūnių bei briaunų skaičiaus grafuose, GRA algoritmų išgaunami rezultatai skiriasi. Deja, šiame darbe neatliekamas pakankamai detalus tyrimas apie grafų struktūros įtaką GRA algoritmams. Tolimesniame darbe galima būtų atlikti detalesnį grafų struktūros tyrimą, siekiant nustatyti, kurie euristiniai grafų redagavimo atstumo algoritmai tinkamiausi tam tikro pobūdžio grafams. Taip pat, galima būtų atlikti daugiau eksperimentų su generuotais retais, tai yra nedidelio briaunų kiekio, grafais.

Literatūra

[ARR15] Zeina Abu-Aisheh, Romain Raveaux ir Jean-Yves Ramel. A graph database repository and performance evaluation metrics for graph edit distance. *International Workshop on Graph-Based Representations in Pattern Recognition*, p.p. 138–147. Springer, 2015.

[BA99] Albert-László Barabási ir Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[Bab16] László Babai. Graph isomorphism in quasipolynomial time. *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, p.p. 684–697, 2016.

[BBB+20] Nicolas Boria, David B Blumenthal, Sébastien Bougleux ir Luc Brun. Improved local search for graph edit distance. *Pattern Recognition Letters*, 129:19–25, 2020.

[BBB18] Nicolas Boria, Sébastien Bougleux ir Luc Brun. Approximating ged using a stochastic generator and multistart ipfp. *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, p.p. 460–469. Springer, 2018.

[BBC+17] Sebastien Bougleux, Luc Brun, Vincenzo Carletti, Pasquale Foggia, Benoit Gaüzère ir Mario Vento. Graph edit distance as a quadratic assignment problem. *Pattern Recognition Letters*, 87:38–46, 2017.

[BBG+19] David B Blumenthal, Sébastien Bougleux, Johann Gamper ir Luc Brun. Gedlib: a c++ library for graph edit distance computation. *International Workshop on Graph-Based Representations in Pattern Recognition*, p.p. 14–24. Springer, 2019.

[BBG+20] David B Blumenthal, Nicolas Boria, Johann Gamper, Sébastien Bougleux ir Luc Brun. Comparing heuristics for graph edit distance computation. *The VLDB Journal*, 29(1):419–458, 2020.

[BDB+18] David B Blumenthal, Evariste Daller, Sébastien Bougleux, Luc Brun ir Johann Gamper. Quasimetric graph edit distance as a compact quadratic assignment problem. *2018 24th International Conference on Pattern Recognition (ICPR)*, p.p. 934–939. IEEE, 2018.

[BG17a] David B Blumenthal ir Johann Gamper. Correcting and speeding-up bounds for non-uniform graph edit distance. *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, p.p. 131–134. IEEE, 2017.

[BG17b] David B Blumenthal ir Johann Gamper. Improved lower bounds for graph edit distance. *IEEE Transactions on Knowledge and Data Engineering*, 30(3):503–516, 2017.

[BGB16] Sébastien Bougleux, Benoit Gaüzere ir Luc Brun. Graph edit distance as a quadratic program. *2016 23rd International Conference on Pattern Recognition (ICPR)*, p.p. 1701–1706. IEEE, 2016.

[BGG+71] Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière ir O Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, 1971.

[BL71] François Bourgeois ir Jean-Claude Lassalle. An extension of the munkres algorithm for the assignment problem to rectangular matrices. *Communications of the ACM*, 14(12):802–804, 1971.

[DBG+18] Évariste Daller, Sébastien Bougleux, Benoit Gaüzère ir Luc Brun. Approximate graph edit distance by several local searches in parallel. 2018.

[FW+56] Marguerite Frank, Philip Wolfe ir k.t. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

[GBR+14] Benoit Gaüzère, Sébastien Bougleux, Kaspar Riesen ir Luc Brun. Approximate graph edit distance guided by bipartite matching of bags of walks. *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, p.p. 73–82. Springer, 2014.

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann ir Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[Hou10] Stefan Hougardy. The floyd–warshall algorithm on graphs with negative cycles. *Information Processing Letters*, 110(8-9):279–281, 2010.

[JH06] Derek Justice ir Alfred Hero. A binary linear programming formulation of the graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1200–1214, 2006.

[Kal15] Sushilkumar Kalmegh. Analysis of weka data mining algorithm reptree, simple cart and randomtree for classification of indian news. *International Journal of Innovative Science, Engineering & Technology*, 2(2):438–446, 2015.

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, p.p. 302–311, 1984.

[Kuh56] Harold W Kuhn. Variants of the hungarian method for assignment problems. *Naval Research Logistics Quarterly*, 3(4):253–258, 1956.

[LAR+17] Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux ir Sébastien Adam. New binary linear programming formulation to compute the graph edit distance. *Pattern Recognition*, 72:254–265, 2017.

[LHS09] Marius Leordeanu, Martial Hebert ir Rahul Sukthankar. An integer projected fixed point method for graph matching and map inference. *Advances in neural information processing systems*, p.p. 1114–1122, 2009.

[Mun57] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[Par11] Nidhi Kiranbhai Parikh. *Generating random graphs with tunable clustering coefficient*. Disertacija, Virginia Tech, 2011.

[Qui14] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[RB08] Kaspar Riesen ir Horst Bunke. Iam graph database repository for graph based pattern recognition and machine learning. *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, p.p. 287–297. Springer, 2008.

[RB09] Kaspar Riesen ir Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision computing*, 27(7):950–959, 2009.

[Rie15] Kaspar Riesen. Structural pattern recognition with graph edit distance. *Advances in computer vision and pattern recognition*. Springer, 2015.

[SF83] Alberto Sanfeliu ir King-Sun Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (3):353–362, 1983.

[STF+17] Michael Stauffer, Thomas Tschachtli, Andreas Fischer ir Kaspar Riesen. A survey on applications of bipartite graph edit distance. *International Workshop on Graph-Based Representations in Pattern Recognition*, p.p. 242–252. Springer, 2017.

[ZTW+09] Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng ir Lizhu Zhou. Comparing stars: on approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, 2009.

[ZZL+14] Weiguo Zheng, Lei Zou, Xiang Lian, Dong Wang ir Dongyan Zhao. Efficient graph similarity search over large graph databases. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):964–978, 2014.

Priedai

Algoritmas 4 Laipsninio dėsnio grafų generavimo algoritmas

procedure PowerGraph(n, m)
  $n_0 = m + 2$
  $V = \{1, 2, \ldots, n_0\}$
  $E = \{(1, n_0)\} \cup \{(i, i+1) \mid 1 \leq i \leq n_0 - 1\}$
  for $i = 1$ **to** n_0 **do**
    $PS[i] = \sum_{k=1}^{i} d(v_i)$
  end for
  for $i = n_0 + 1$ **to** n **do**
    $V = V \cup \{i\}$
    $e = 0$
    while $e < m$ **do**
      $r =$ *RandomNumber(1, PS[i - 1])*
      $v =$ *FindNode*(r, PS)
      if $(i, v) \notin E$ **then**
        $E = E \cup \{(i, v)\}$
        $e = e + 1$
        for $j = v$ **to** i **do**
          $PS[j] = \sum_{k=1}^{i} d(v_i)$
        end for
      end if
    end while
  end for
  return (V, E)
end procedure
procedure FindNode(r, PS)
  $low = 1$
  $high = sizeof PS$
  while $low < high$ **do**
    $mid = \frac{high+low}{2}$
    if $r \leq PS[low]$ **then**
      return low
    else if $low = mid$ **then**
      return $high$
    else if $r \leq PS[mid]$ **then**
      $high = mid$
    else
      $low = mid$
    end if
  end while
  return low
end procedure