VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Originalumo paieškos euristikos pritaikymas ir palyginimas grafų optimizavimo uždaviniuose

Novelty search application and comparison in graph-based optimization problem domain

Magistro baigiamasis darbas

Atliko:	Laimonas Beniušis	(parašas)
Darbo vadovas:	dr. Julius Žilinskas	(parašas)
Recenzentas:	asist. dr. Adomas Birštunas	(parašas)

Vilnius – 2020

Santrauka

Grafų optimizavimo užduotys kaip keliaujančio pirklio ir grafo spalvinimo turi didžiulį praktinį pritaikymą bei yra plačiai naudojamos matuoti euristinių algoritmų efektyvumą. Inovatyvi idėja, kuri gimė visai kitoje kompiuterių mokslo srityje, atsisakyti tikslo funkcijos grafų optimizavimo uždaviniuose buvo praktiškai įgyvendinta keliaujančio pirklio ir grafų spalvinimo atvejuose.

Darbe yra plačiai aptarta grafų anomalijų paieška, kuri tiesiogiai susijusi su originalumo paieškos idėja, nes anomalijų paieška yra natūralus žingsnis link naujų panašumo funkcijų aptikimo. Panašumo funkcijos yra pagrindinis variklis originalumo paieškoje, tačiau taip pat yra vienintelis negriežtai apibrėžtas dalykas. Dėl panašumo funkcijos įvairovės ir nenuspėjamumo, jos efektyvumo negalima žinoti jo nepamatavus. Šioje vietoje susiduria kūryba su mokslu, kas dar skatina eksperimentavimą su neįprastomis panašumo funkcijomis. Konkretizuota **originalumo paieškos euristika nėra efektyvi** bendro grafų optimizavimo uždavinio atveju, tačiau yra pranaši (lyginant su tradicinėmis euristikomis) tokiu atveju, kada konstruojamas iteratyviai sunkiai pagerinamas grafas.

Darbe buvo palyginamas įvairių euristinių algoritmų efektyvumas bei jų parametrų kaitos įtaka. Pasiūlyta ir pritaikyta metodika rasti pakankamai gerą parametrų kombinaciją, kuri buvo naudojama palyginti algoritmų efektyvumui tarpusavyje beveik lygiose sąlygose. Darbe pateikiamos kelios panašumo funkcijos, kurios buvo naudojamos originalumo paieškos euristikos tyrimo metu, kaip ir panašumo funkcijos pritaikytos iš anomalijų aptikimo. Be to, darbe yra pateikiamas naujas sprendinių archyvavimo metodas, kuris simuliuoja hierarchinį negriežtą klasterizavimą, kuris grupuoja sprendinius pagal panašumą pagreitinantis optimizavimo eigą bei suteikia archyvui struktūros. Taip pat pasiūlytos tradicinių euristinių algoritmų modifikacijos.

Raktiniai žodžiai: Euristiniai algoritmai, keliaujančio pirklio uždavinys, grafo spalvinimo uždavinys, viršūnių spalvinimas, modeliuotas atkaitinimas, skruzdžių kolonijos sistema, anomalija, optimizavimas, nedeterministinis, grafas, įvairovė, betikslis optimizavimas, originalumo paieška, beribis.

Summary

Travelling salesman problem and graph coloring problem is widely known as the most applicable and used as a benchmark to measure the effectiveness of various heuristic algorithms. innovative idea to do away with a goal function (which originated in a whole widely different computer science field) was formed into a concrete implementation and applied to travelling salesman problem and graph coloring problem domain.

In this thesis the search for anomalies in graph domain was extensively discussed as a valuable bridge to the idea of novelty search. The search for anomalies can help form new similarity functions, which are the main driving force behind novelty search. Moreover, it is the only loosely defined concept, which in turn results in its unpredictability. This is the point where creativity and science meet, which encourages even more experimentation with unusual similarity functions. Formalized **novelty search is not effective in a general graph optimization case** but is superior (compared with traditional heuristics) in specific case where graph is intentionally constructed to be hard to improve iteratively.

In this thesis there was an experiment regarding the efficiency of traditional heuristic algorithms and the parameter influence of said algorithms. Furthermore, presented methodology of combining parameters was used to find a good enough combination of parameters which in turn lead to pretty fair comparison of the effectiveness of heuristic algorithms. Moreover, in this thesis there are several descriptions of similarity functions all of which are used during the novelty search experiment besides the similarity functions derived from various anomaly search algorithms. Also, a brand new archiving method is proposed in this thesis which simulates hierarchical fuzzy clustering by grouping similar solutions thus increasing the speed of the algorithm and providing a structure to the archive. Lastly, there are several modifications to the traditional heuristic algorithms proposed in this thesis.

Keywords: heuristic, travelling salesman problem, TSP, graph coloring problem, GCP, vertex coloring, simulated annealing, ant colony system, anomaly, optimization, nondeterministic, graph, diversity, goalless optimization, novelty search, open-ended.

Turinys

1. Įvadas 5
   1.1. Originalumo paieškos idėjos pirmtakas 5
   1.2. Originalumo paieškos formalizavimas ir empirinis patvirtinimas 7
   1.3. Darbo mokslinis naujumas 9
   1.4. Darbo rezultatų praktinė reikšmė 9
2. Originalumo paieškos pradmenys evoliuciniuose skaičiavimuose 10
   2.1. Įvairovės palaikymas 10
   2.2. Daugiakriterinis optimizavimas palaikant įvairovę 10
   2.3. Originalumo paieškos euristika 11
3. Originalumo aptikimas grafuose 13
4. Originalumo aptikimas viename dideliame grafe 14
   4.1. Viršūnių originalumo aptikimas 14
   4.2. Pografių originalumas 15
   4.3. Originalumo nustatymas kintančiuose grafuose 16
      4.3.1. Anomalijų aptikimas kaimynysčių evoliucijos metu 16
         4.3.1.1. Klasterių palaikymas ir evoliucinė analizė 17
         4.3.1.2. Kaimynystės analizė 18
      4.3.2. Trumpiausio kelio atstumu paremtas anomalijų aptikimas 18
   4.4. Originalumo aptikimo grafuose santrauka 20
5. Tradiciniai euristiniai algoritmai grafų optimizavimo uždaviniams 21
   5.1. Atkaitinimo modeliavimas 21
   5.2. Skruzdžių kolonijos sistema 22
   5.3. Genetinis algoritmas 23
6. Originalumo paieška ir jos variacijos 25
   6.1. Originalumo paieškos idėjos konkretizavimas 25
      6.1.1. Originalaus elgesio atstumo funkcija 25
7. Keliaujančio pirklio uždavinio eksperimentų sąlygų konkretizavimas 26
   7.1. Panašumo funkcijos 26
      7.1.1. Tikslo funkcijos įverčio nuotolis 26
      7.1.2. Ilgiausia bendra viršūnių seka 26
      7.1.3. Sutampančių lankų kiekis 26
   7.2. Sprendinio lokalūs pokyčiai (mutacijos) 27
      7.2.1. Viršūnės perkėlimas 27
      7.2.2. Dviejų viršūnių apkeitimo mutacija 27
      7.2.3. Kelio dalies nukirtimo ir regeneravimo mutacija 27
      7.2.4. Centrinės inversijos mutacija 28
      7.2.5. Vidinės sekos apvertimo mutacija 28
   7.3. Genetinio algoritmo pritaikymas 28
      7.3.1. Dalinai perduodamas kryžminimas 28
      7.3.2. Eilės numerio kryžminimas 29
      7.3.3. Ciklinis kryžminimas 30
      7.3.4. Implementacijos detalės 31
   7.4. Atkaitinimo modeliavimo pritaikymas 31
   7.5. Skruzdžių kolonijos sistemos algoritmo pritaikymas 31

8. Grafo viršūnių spalvinimo uždavinio eksperimentų sąlygų konkretizavimas 32
8.1. Grafo reprezentacijos papildas 32
8.2. Panašumo funkcijos 33
8.2.1. Tikslo funkcijos įverčio nuotolis 33
8.2.2. Vienodų spalvų pozicijos 33
8.2.3. Vienodas izomorfiškumas 33
8.3. Sprendinio lokalūs pokyčiai 34
8.3.1. Godžioji euristika DSatur 34
8.3.2. Nelegalios mutacijos ištaisymas 34
8.4. Genetinio algoritmo pritaikymas 35
8.4.1. Skaidinių kryžminimas 35
8.4.2. Pografius atskiriantis kryžminimas 36
8.4.3. Vieno atkirtos taško kryžminimas 36
8.4.4. Dviejų atkirtos taškų kryžminimas 37
8.4.5. Implementacijos detalės 37
8.5. Atkaitinimo modeliavimo pritaikymas 37
8.6. Skruzdžių kolonijos sistemos algoritmo pritaikymas 37
9. Originalumo paieškos pritaikymas 39
9.1. Hierarchinis sprendinių panašumo palyginimas 39
10. Sunkiai sprendžiamų keliaujančio uždavinių konstravimas 41
11. Parametrų lyginimo schema ir rezultatų formavimas 43
11.1. Rezultatų formavimas 44
12. Rezultatai 45
12.1. Palankiausios algoritmų aplinkos 45
12.1.1. Genetinis algoritmas (GA) 45
12.1.2. Skruzdžių kolonijos sistema (ACO) 45
12.1.3. Atkaitinimo modeliavimas (SA) 45
12.1.4. Originalumo paieška (NS) 46
12.2. Konkrečių uždavinių ir algoritmų rezultatai 46
13. Išvados 49
Literatūra 51

1. Įvadas

1.1. Originalumo paieškos idėjos pirmtakas

Kodėl sunku pasiekti ambicingą tikslą? Norint pradėti spręsti šią problemą, pagalvokime kiek gali būti įmanomų dalykų, konkrečiau - visus įmanomus vaizdus. Viskas, ką esame matę, dar matysim ir ko niekada nematysim yra šioje kategorijoje. Labai maža proporcija šių vaizdų yra tokie meno šedevrai kaip Leonardo da Vinčio - „Mona Lisa“ ar Vincento van Gogo - „Žvaigždėtas dangus“. Tai yra sunkiai pasiekiami tikslai. Žymiai didesnę dalį sudaro vaizdai kurie yra atpažįstami, tačiau mažiau įkvepiantys nei minėti šedevrai, kaip pažįstamų žmonių veidai ar kasdieniniai objektai. Žinoma, didžioji dalis vaizdų yra beverčiai, panašūs į vaizdą kai yra blogas televizijos ryšis - elektrinės statikos triukšmą. Tačiau, viena gera savybė tokios didžiulės aibės vaizdų, kad nemažas kiekis šedevrų dar neegzistuoja, nes niekas jų dar nenutapė ar nenupiešė, arba kitaip tariant, šie ateities šedevrai dar nebuvo atrasti.

Galima galvoti apie tikslo siekimą kaip atradimo procesą. Šedevro tapymas yra iš esmės jo atradimas tarp visų įmanomų vaizdų, tarsi paieška šioje aibėje. Savaime suprantama, tokia paieška žymiai skiriasi nuo, pavyzdžiui, kojinės paieškos skalbimo mašinoje. Tokia vaizdo paieška yra panaši į menininko atliekamą intuicinį idėjos išplėtimą. Viso šito esmė yra ta, kad tokią, menininko atliekamą, paiešką galima būtų pritaikyti ir labiau pragmatiškose srityse kaip mokslo ar technologijų, kas galėtų būti naujos teorijos ar išradimai. Kad ir koks bebūtų tikslas, jo siekimo būdas yra daugiau ar mažiau panašus į atradimo procesą - iš daugybės galimybių norime rasti sau tinkamą.

Taigi, galime apie kūrybingumą kabėti kaip apie tam tikrą paiešką. Vienas iš paieškos komponentų yra paieškos erdvė - visų įmanomų dalykų aibė. Pabandykime įsivaizduoti tokią paieškos erdvę kaip labai didelį kambarį, kuriame skirtingose vietose yra išdėlioti skirtingi vaizdai. Šiame kambaryje vaizdai turės tam tikrą tvarką, pavyzdžiui, viename kampe yra visi veidai, kitame visos „Mona Liza“ variacijos (tarp jų ir Leonardo da Vinčio šedevras). Kadangi dauguma vaizdų yra panašūs į televizijos elektrinės statikos triukšmą, rasti kažką gražaus ar kas mus domintų yra sunku. Kalbėti apie kūrybą, kaip paiešką tokiame kambaryje, yra prasminga, nes ką nutapys menininkas priklauso nuo to, kokias kambario dalis jau matė. Pavyzdžiui, jeigu nėra matęs pointilizmo (tapybos technika, kada tapoma mažais taškeliais ar potėpiais), tai mažai tikėtina, kad šią techniką staiga pats išras. Tam tikra prasme, civilizacija šį kambarį nagrinėja jau labai senai, todėl kuo daugiau šio kambario išnagrinėjame, tuo labiau kartu, kaip civilizacija, suprantame kas yra įmanoma.

Tarkime, menininko tikslas - nutapyti peizažą. Jeigu jis yra patyręs peizažų tapytojas, vadinasi jis yra buvęs toje kambario vietoje su peizažais. Iš tos kambario vietos galima išsišakoti į tas, kurios yra pilnos dar neįsivaizduotų peizažų, tačiau jei menininkas nėra susipažinęs su peizažo tipu, nėra tikėtina, kad jis nutapys šedevrą, net jeigu jo toks tikslas. Tam tikra prasme, visos mūsų aplankytos vietos, ar tai mintyse, ar gyvai, yra laipteliai naujoms idėjoms. Toks mąstymo būdas tinka ne tik su vaizdais. Galima įsivaizduoti kambarį pilną bet ko, pavyzdžiui - išradimų. Analogiškai kaip ir su vaizdais, dauguma išradimų yra beprasmiški menkniekiai, bet geriau pa-

ieškojus būtų galima rasti paprastų, tačiau be galo naudingų išradimų, tokių kaip ratas, krepšys ar ietis. Dar geriau paieškojus, galima būtų rasti žymiai sudėtingesnių išradimų kaip automobiliai ar kompiuteriai, o dar kitur slypi išradimai, laukiantys kol juos kažkas suras. Jeigu pažiūrėtume į kompiuterių kampą - pirmasis kompiuteris buvo ENIAC, išrastas 1946, kuris atliko 5000 instrukcijų per sekundę [GG09]. Palyginus su moderniais kompiuteriais, tai milijoną kartų mažiau. Tada kyla klausimas, tai jeigu modernus kompiuteris yra tarp tos paieškos erdvės variantų, kodėl pradėti nuo tokio lėto? Prieš išrandant kompiuterį, niekas nes nežinojo, kad tai yra įmanoma - niekas nebuvo toje kambario pusėje su kompiuteriais. Kitaip sakant, nebuvo aiškių laiptelių, kurie nuvestų prie pirmojo kompiuterio, kaip ir dabar nėra aiškių laiptelių kurie nuvestų prie dar milijoną kartų greitesnio kompiuterio. Tokie laipteliai yra aukštesnio galimybių lygio kelio dalis. Kol nėra pasiekti visi laipteliai, negali būti pasiektas ir pagrindinis tikslas.

Pagrindinė tokių pamąstymų išvada yra ta, kad tikslas turi būti realistiškas, tikintis jį pasiekti. Įsivaizduojamo didelio kambario metafora, kurio erdvėje vyksta paieška, padeda laiptelių principą pateisinti. Norint pasiekti ambicingų tikslų pirmiausia reikia išsiaiškinti ar galima surasti kelią tarp šių laiptelių kambaryje, nuo dabar esamos vietos iki norimos būtį ateityje. Tikslas gali būti naudingas, jeigu jis yra pakankamai realistiškas, tačiau kada jis yra ambicingas - viskas drastiškai pasikeičia, iki tiek, kad vien tik tikslo išsikėlimas gali tapti kliūtimi jo pasiekimui, ypač tokiose srityse kurios reikalauja išradingumo, kūrybingumo ar novatoriškumo. Taigi, išplaukia šioks toks paradoksas, kuris teigia, kad pasiekimai tampa sunkiau pasiektini, kada yra paverčiami tikslais. Šis paradoksas veda prie labai keistos išvados - geriausias būdas pasiekti didžių dalykų ar neribotų ambicijų yra neturėti tikslo. Kaipgi taip gali būti? Atsakymas slypi tuose laipteliuose, kurie sudaro kelią link tikslo. Šie laipteliai dažniausiai būna labai keisti, kuriuos sunku nuspėti, jeigu būtų galvojama tik apie tikslą. Istoriškai gausu tokių atvejų. Pavyzdžiui, pats pirmasis kompiuteris buvo sukurtas naudojant vakuumines lempas, kurių raida neturi nieko bendra su kompiuteriais. Tomas Edisonas jas nagrinėjo dėl elektros, o ne kompiuterijos. Vėliau, 1904, fizikas John A. Fleming šią technologiją pritaikė radijo bangų aptikimui - vis dar nematyti kompiuterijos. Tik po kelių dešimtmečių mokslininkai susivokė, kad jas galima būtų panaudoti kuriant kompiuterius, kada ENIAC (pirmasis kompiuteris) buvo galiausiai sukurtas.

Taigi, nors ir vakuumo lempos yra pagrindinis laiptelis kompiuterių sukūrimo kelyje, beveik niekas galėjo to numatyti. Net jeigu kažkoks mokslininkas, gyvenęs 18 a., turėtų tikslą išrasti kompiuterį, jis niekad nepagalvotų, kad galbūt pirmiausiai reikėtų sukurti vakuuminę lempą. Pagrindinė problema slypi tame, kad laipteliai neprimena galutinių tikslų. Vakuuminės lempos savaime žmonėms neprimena kompiuterio, tačiau pagal taip, kaip išsirutuliavo istorija, kompiuteriai išradimų kambaryje būtų šalia vakuuminių lempų. Kada jau turi vakuumines lempas - jau beveik turi ir pirmą kompiuterį. Tame ir pasireiškia problema, kad beveik niekas negali to numatyti iš anksto - paieškos erdvės tvarka yra visiškai nenuspėjama.

Toks nenuspėjamumas yra labiau kaip taisyklė nei išimtis siekiant beveik visų ambicingų tikslų. Jeigu kažkieno tikslas buvo sukurti mikrobangų krosnelę, tai jis nebūtų dirbęs su radarais, jeigu sukurti skraidančią mašiną - nebūtų praleidęs ateinančius dešimtmečius kuriant motorą ir jeigu norėtų sukurti pirmąjį kompiuterį - nebūtų tobulinęs vakuuminių lempų. Laipteliai buvo

tobulinami tų žmonių, kurie neturėjo galutinio tikslo sukurti mikrobangų krosnelę, lėktuvą ar kompiuterį. Šio išradimų kambario struktūra yra tokia keista, kad galutinio tikslo siekimas gali atitraukti nuo jam pasiekti reikalingų laiptelių, kitaip tariant, jeigu per daug galvosi kaip sukurti kompiuterį tai niekad nepagalvosi apie vakuumines lempas. Taigi, ambicingi tikslai yra dažnai apgaulingi. Suplanuotas kelias iki tikslo neretai nepataiko ant reikalingų laiptelių. Dėl šios priežasties, tikslo funkcija, kuri matuoja ar judame tikslinga kryptimi, dažnai yra apgaulinga, nes ji ignoruoja laiptelius, kurie yra būtini tikslo pasiekimui.
Taip idėjos radikalumą pateisina autoriai Kenneth O. Stanley ir Joel Lehman [SL15]. Šios idėjos esmė yra sukurti tokį paieškos modelį, kuris remiasi panašia paieška kurią atlieka menininkas ieškodamas idėjų. Ši idėja gimė iš projekto (kuriam vadovavo Kenneth O. Stanley) pavadinimu Picbreeder [SBD+08].

1.2. Originalumo paieškos formalizavimas ir empirinis patvirtinimas

Daugybė svarbių užduočių, kada yra bandomos spręsti euristiniais optimizavimo metodas, dažniausiai naudojant kokią nors tikslo funkcija, pasirodo esą apgaulingos. Tai reiškia, kad mažas postūmis link tikslo nebūtinai lemia bendrą progresą. Ši problema dar yra žinoma kaip lokalus ekstremumas. Optimizavimo algoritmai šią problemą bando spręsti įvedami šiek tiek atsitiktinumo ar įvairovės nuo optimumo taško. Pavyzdžiui atkaitinimo modeliavime (angl. *simulated annealing*) yra įvedama tikimybė pasirinkti blogesnį tarpinį sprendinį (eiti tolyn nuo globalaus optimumo), siekiant išsikapstyti iš lokalaus minimumo taško. Šis ir panašūs euristiniai algoritmai įveda papildomus žingsnius, kurie skirti lokalaus optimumo problemai išspręsti, tačiau vis tiek paieška yra paremta specifine tikslo funkcija. Netgi evoliuciniuose skaičiavimuose, kur yra pilna atsitiktinumo (mutacijos ir kryžminimas), atranka yra paremta tikslo funkcija, kas iš esmės lemia tos pačios problemos neišvengiamumą.

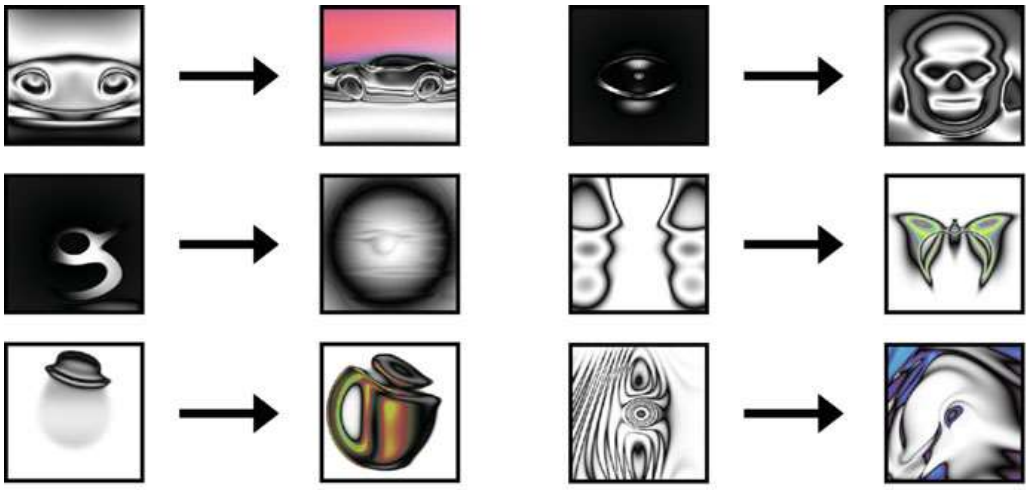
Originalumo paieška (angl. *Novelty search*) (toliau NS), yra evoliuciniais skaičiavimais paremta paieška, kuri iš viso atsisako konkrečios tikslo funkcijos, bet bando modeliuoti smalsumą [LEH12], arba kitaip tariant, ieško to, kas yra įdomu. Pilnai formalizuoti, kas gali būti įdomu, yra neįmanoma, tačiau gera pradžia yra pradėti paiešką nuo kažko naujo, neįprasto - originalaus. Ši idėja yra paremta klaidžiojimu po visų įmanomų dalykų kambarį, aptinkant laiptelius, kurie dar yra nematyti ir kurie galbūt padės aptikti galutinį tikslą. NS algoritmas dar yra vadinamas kaip tokių laiptelių surinkėjas (angl. *stepping stone collector*) [SL15]. Vieni laipteliai gali nuvesti prie kitų laiptelių ir taip toliai, kol galiausiai visi pagrindiniai laipteliai bus aptikti ir galutinis tikslas taps pasiekiamas.

Ankščiau minėtame projekte Picbreeder, buvo pritaikytas pradinis NS algoritmo variantas. Tai yra tinklalapis, kur žmonės gali gaminti naujus paveikslėlius mutacijų būdu - pasirinkto paveikslėlio vaikai turi tėvinio paveikslėlio bruožus su mažomis modifikacijomis. Šiuo atveju, tėvų pasirinkimas nebuvo automatizuotas - žmonės pasirinko pagal įdomumą. Įdomius paveikslėlius vartotojai gali paviešinti, suteikti pavadinimą ar kategoriją. Kiti vartotojai gali pradėti paveikslėlio evoliuciją ne nuo pradžių, bet nuo kažkieno jau paviešinto paveikslėlio. Tokiu principu ir buvo rasti patys įdomiausi paveikslėliai kaip pavyzdžiui 1 pav.

Patys paveikslėliai nebuvo keičiami, bet generuojami vidinio kompozicinio ornamentus gaminančio tinklo pagalba (angl. *Compositional pattern producing network*) [Sta07]. Tai yra dirbtinio neuroninio tinklo variacija, kuri pasižymi skirtingų aktyvacijos funkcijų kompozicija. Šis tinklas kiekvienam normalizuotam pikseliui suteikia spalvos reikšmę, kas ir pagamina paveikslėlį. Visi Picbreeder vartotojų paviešinti paveikslėliai yra viešai prieinami.

1 pav. Vienas iš populiariausių Picbreeder surastų paveikslėlių „Drugelis“

Picbreeder projektas taip pat pateisina dar vieną įžangoje minėtą idėja - tarpiniai žingsniai link tikslo dažnai jo neprimena. Tai pasireiškia Picbreeder projekte, kai rasti paveikslėliai visai neprimena savo pirmtakų, tačiau jie buvo būtini galutinio paveikslėlio pasiekimui, kas ir yra parodoma 2 pav.



2 pav. Picbreeder projekte evoliucionuoti paveikslėliai, kurie visai neprimena savo pirmtakų.

Picbreeder eksperimentas buvo pakartotas su tikslo funkcija. Tikslas buvo pakartotinai, tačiau šį kartą automatizuotai surasti paveikslėlius tokius kaip 1 pav. Šis eksperimentas nepasiteisino, nes po 30000 generacijų gauti paveikslėliai neprilygo tiems, kuriuos rado žmonės vedami smalsumo (kurių dauguma neviršijo šimto generacijų). Picbreeder projektas epiriškai patvirtino tokio tikslu nesiremiančio metodo prasmingumą.

1.3. Darbo mokslinis naujumas

Tradiciniai euristiniai algoritmai pasižymi vienoda savybe – tikslo funkcija. Be išimties yra daroma prielaida, kad progresą įmanoma išmatuoti ir jį gerinti pastoviai. Sunkūs uždaviniai pasižymi tuo, kad pastovus tikslo funkcijos optimizavimas nepriveda link globalaus optimumo, todėl tradiciniai euristiniai algoritmai dažnai nepajėgia jų išspręsti (ar net rasti arti sprendinio artimo optimaliam). Tam spręsti buvo sugalvota originalumo paieškos idėja, kuri teigia ieškoti kažko naujo, originalaus, dar nematyto. Ši idėja buvo pritaikyta dirbtinių neuroninių tinklų evoliucijos kontekste, kai sprendinio reprezentacija ir jo duotas sprendinys yra ne tas pats. Tai yra, dirbtinis neuroninis tinklas ir juo generuojamas rezultatas nėra vienas ir tas pats. Problema iškyla, kada sprendinys ir jo rezultatas yra vienas ir tas pats. Tokia problema yra grafų optimizavimo uždaviniuose kaip keliaujančio pirklio bei grafo spalvinimo. Šie uždaviniai yra pakankamai skirtingi, todėl yra galima palyginti originalumo paieškos efektyvumą skirtingose aplinkose. Keliaujančio pirklio uždavinio atveju yra naudojami pilni grafai bei akcentuojamas lanko svoris, o grafo spalvinimo uždavinio atveju yra naudojami įvairūs grafai bei akcentuojamas viršūnių jungumas. Taigi, galutinis darbo tikslas yra originalumo paieškos efektyvumą palyginti su tradiciniais euristiniai algoritmais sprendžiant minėtus uždavinius.

1.4. Darbo rezultatų praktinė reikšmė

Inovatyvi idėja praktiškai pritaikyta seniai žinomuose grafų uždaviniuose. Originalumo paieška yra neturinti nustatytų ribų (angl. *open-ended*) euristika. Šiame darbe ištirtiems uždaviniams tokia euristika nėra efektyvi, arba reikalauja per daug resursų, kad būtų pateisinama. Yra pagrindžiama, kad tokia ir panaši technika netinkama šiems uždaviniams spręsti, išskyrus kada yra konkretus atvejis, kuris specialiai blokuoja iteratyvius sprendinių pagerinimus.

2. Originalumo paieškos pradmenys evoliuciniuose skaičiavimuose

Originalumo įvertis yra tam tikro tipo tikslo funkcija, dėl to, norint gauti primityvią originalumo paieškos variaciją, tereikia adaptuoti esamą euristinį algoritmą, pakeičiant jo tikslo funkciją. Tikslo funkcijos įvertis yra pajėgumas (angl. *fitness*). NS algoritmo pagrindinis panaudojimas yra evoliuciniais skaičiavimais paremtu algoritmu [LS08a], skaičiuojant atstumą tarp sprendinių. Iš pirmo žvilgsnio, tai mažai kuo skiriasi nuo įvairovę didinančios (arba sprendinių panašumą mažinančios) charakteristikos, tačiau visi algoritmai, kurie pasinaudoja įvairovės didinimu, neapsieina be konkrečios tikslo funkcijos. Kitaip tariant, įvairovė nėra pagrindinis tikslas.

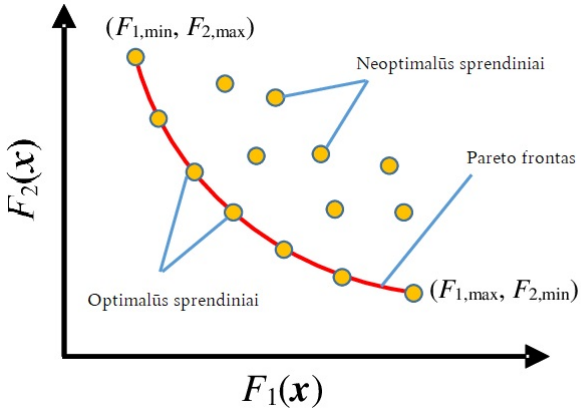
2.1. Įvairovės palaikymas

Populiarus būdas išvengti per daug ankstyvaus konvergavimo evoliuciniuose skaičiavimuose yra įvairovės palaikymas. Daugybė tokių metodų yra paremti individų (sprendinys, kuris dalyvauja evoliucijoje) išskirstymu į nišas (rases) [GR$^+$87; HL06; Mah95], kuriose vyksta individų varžybos, o ne visoje individų populiacijoje (sprendinių aibė). Pagrindinis tokio įvairovės palaikymo privalumų yra toks, kad jeigu viena individų niša įklimpo į lokalų optimumą, yra tikimybė, kad kita, pakankamai skirtinga, rasė to nepadarys. Pavyzdžiui, pajėgumo dalinimosi (angl. *fitness sharing*)[GR$^+$87] metodas skatina sprendimų radimą tolimose paieškos erdvės vietose. Kitaip tariant, sprendiniai, kurie yra tankiuose paieškos erdvės regionuose, bus mažesnio pajėgumo.

Kitas įvairovės palaikymo metodas yra pajėgumo uniforminis pasirinkimas (angl. *fitness uniform selection*), kuris atsisako pajėgumo didinimo skatinimo [HL06]. Tokiu atveju, evoliucinė individų atranka (angl. *selection*) bando išlaikyti uniforminę pajėgumo reikšmių distribuciją. Tokiu būdu yra skatinamos unikalios pajėgumo reikšmės (lyginant su visa kita populiacija). Toks metodas yra iš principo kitoks nei dauguma kitų, skirtų įvairovei didinti. Visgi, panašiai įvertintų, tačiau ženklai skirtingų sprendinių apjungimas gali būti ir nenaudingas bendram progresui [LEH12].

2.2. Daugiakriterinis optimizavimas palaikant įvairovę

Įvairovės charakteristiką galima formalizuoti kaip vieną iš tikslų daugiakriteriniuose evoliucinių skaičiavimų algoritmuose (angl. *Multi-objective evolutionary algorithms*) [Lyn07]. Tokie algoritmai optimizuoja kelias tikslo funkcijas iš karto. Pagrindinis skirtumas nuo vienatikslių optimizavimo algoritmų yra sprendinių gerumo formuluotė. Du skirtingi sprendiniai gali būti vienas už kitą geresni skirtingų tikslų kontekstuose, todėl tarpusavyje nedominuojantys (vienas nėra visapusiškai geresnis už kitą). Tokių tarpusavyje nepalygintinų sprendinių aibė vadinama Pareto frontu (angl. *Pareto-front*). Pavyzdžiui, jeigu turime dviejų tikslų daugiakriterinio optimizavimo rezultatą (sprendinių aibė), jo Pareto frontas galėtų atrodyti taip: (3 pav.).



3 pav. Pareto fronto pavyzdys dviejų tikslo funkcijų atveju. F_1,F_2 - optimizuojamos tikslo funkcijos. Pareto fronto aibėje yra optimalūs sprendiniai, kurie yra tarpusavyje nepalyginami

Bendru atveju, daugiatikslės optimizavimo funkcijos nėra apsaugotos nuo užduoties apgaulingumo problemos [Deb99]. Jos tik papildo optimizavimo paieškos kryptis. Kitaip tariant, jeigu paieška įklimpo lokaliame optimume vieno iš tikslo funkcijų atžvilgiu, yra tikėtina, kad bendras progresas pajudės kitų tikslo funkcijų atžvilgiu. Siekiant išvengti apgaulingumo, vienatikslę optimizavimo užduotį galima praplėsti iki daugiatikslės tačiau, kaip ir daugumoje kitų optimizavimo algoritmų, tikslo funkcijos išlaikymas neišvengiamai nuves prie lokalaus optimumo problemos.

2.3. Originalumo paieškos euristika

Originalumo paieškos euristiką galima apibrėžti kaip euristinį optimizavimo algoritmą (šiuo atveju, optimizavimo algoritmas yra evoliucinis), kurio tikslo funkcija remiasi originalumo įvertčiu [LEH12]. Tai yra kažkiek panašu į įvairovės skatinimą, tačiau iš esmės skiriasi. Originalumas, tam tikra prasme, yra reliatyvus, nes jis priklauso nuo padaryto progreso. Evoliucinio algoritmo pradžioje, labai prasto pajėgumo (pagal pašalinę tikslo funkciją) individas prilygs žymiai geresnio pajėgumo individui, nes jie abu yra nauji, dar nematyti, todėl įgaus aukštą originalumo įvertį. Vėliau visi paprasti (aptikti atlikus nedaug modifikacijų nuo pradinio sprendinio) sprendiniai bus atrasti. Tada surasti kažką naujo ir paprasto bus vis sudėtingiau, todėl bus skatinami sprendiniai, kurie nėra paprasti arba yra mažai išnagrinėti. Tokiu būdu yra tikimasi, kad algoritmas modeliuos individų pastovų sudėtingėjimą, kuris ir vyksta natūralioje evoliucijoje. Pagrindinis skirtumas nuo įvairovės skatinimo yra toks, kad reikia galėti įvertinti naujų individų originalumą nuo evoliucijos pradžios, tuo tarpu įvairovės skatinimas siekia išlaikyti plataus spektro individų aibę evoliucijos metu. Tankūs paieškos erdvės regionai (sprendinių atžvilgių), yra mažiau originalūs, todėl ir mažiau skatinami. Tada originalumo įvertį galime apibrėžti kaip paieškos erdvės regiono tuštumą (bendru atveju):

$$O(x) = \frac{1}{k}\sum_{i=0}^{k} dist(x,\mu_i) \tag{1}$$

Čia k yra fiksuotas parametras, nustatomas eksperimentiniu būdu, μ_i yra i-asis arčiausias sprendinio x kaimynas, pagal tam tikro atstumo matą $dist$, kuris priklauso nuo užduoties ir duomenų

tipo ir žymį dviejų kaimyninių sprendinių atstumą paieškos erdvėje. Kaimyninių atstumų apskaičiavimas neapsiriboja dabartinę individų populiaciją, tačiau tikrina ir originalių individų archyvą (visos evoliucijos metu sukaupti iš esmės skirtingi individai). Svarbu paminėti, kad paieškos erdvė nėra tyrinėjama tikslingai, nes nėra žinoma, kaip pasiekti tuščias paieškos erdvės vietas, taip pat kaip nėra žinoma, kaip sukonstruoti sprendinį, kuris butų arti globalaus optimumo.

Jeigu naujo individo originalumo įvertis yra pakankamai didelis (viršija tam tikrą ribą O_{min}), tai toks individas yra išsaugojamas originalumo archyve. Originalumo išsaugojimo riba O_{min} kinta evoliucijos metu, priklausomai nuo naujų ir originalių individų radimo dažnio. Jeigu per daug individų pateko į originalumo archyvą, tai O_{min} yra padidinama, jeigu archyvas ilgai nebuvo praplėstas, tai sumažinama.

Originalumo paieška tuo išsiskiria, kad skatina tik individų įvairovę, neatsižvelgiant į jokią kitą tikslo funkciją. Dėl to yra neįmanoma įklimpti lokaliame optimume, nes algoritmas to nematuoja. Taip pat, galima įžvelgti panašumų su atsitiktinio ėjimo algoritmu (angl. *randomized walk*), tačiau tai akivaizdžiai skiriasi. Originalumo paieška kaupia archyvą, kuris rodo kurios paieškos erdvės vietos (sprendiniai), jau buvo aplankytos, tuo tarpu atsitiktinio ėjimo metu aptikti sprendiniai gali kartotis. Vis dėl to, verta pripažinti, kad tikslo ignoravimas jokiu būdu negarantuoja jo pasiekimo.

Tikriausiai susidarė įspūdis, kad originalumo paieška, tai tik tam tikro optimizavimo algoritmo pajėgumo funkcijos pakeitimas į originalumo funkcijos įvertį, tačiau tai ideologiniu požiūriu skiriasi. Pajėgumo maksimizavimas yra daromas siekiant priartinti paiešką prie tam tikro tikslo (paieškos erdvės regiono), o originalumo paieška vyksta neturint supratimo kada ji turi baigtis ir kokia bendra linkme turėtų judėti. Originalumas yra reliatyvus įvertis, kuris yra apibrėžtas tik konkrečios paieškos kontekste. Kitaip tariant, sprendinio originalumas negali būti palygintas skirtingose paieškose ar netgi tarp skirtingų generacijų tos pačios paieškos kontekste, ką akivaizdžiai galime padaryti turint standartinė tikslo funkcija.

3. Originalumo aptikimas grafuose

Vienas iš originalumo požymių yra kažkas iš esmės kitaip. Dažnai toks apibrėžimas būna pritaikomas pašalinių objektų ar anomalijų atpažinimui (angl. *Outlier Detection*). Grafai yra viena iš labai galingų ir bendrų duomenų reprezentacijos formų. Šios struktūros gali būti naudojamos labai skirtingo spektro duomenis pavaizduoti kaip Internetas, socialiniai tinklai, logistiniai susisiekimo tinklai, bei biologiniams ar cheminiams ryšiams. Tokie grafai dažniausiai pasiskirto į dvi grupes [CAg16]:

- Duomenys iš daugybės mažų grafų. Tarp tokių atvejų pavyzdžių yra cheminių ar biologinių elementų kompozicijose. Tie maži grafai gali pasikartoti tarp skirtingų objektų ar tame pačiame objekte. Viršūnių pasikartojimas, dažnai vedantis prie grafų izomorfizmo problemos. Šios grupės grafai nebus nagrinėjami.

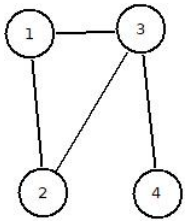
- Duomenys iš vieno didelio grafo. Tarp tokių atvejų pavyzdžių yra Internetas, socialiniai tinklai. Dažniausiai, kiekviena viršūnė atitinka unikalų identifikatorių (pvz. URL Internete). Tokiais atvejais originalumo įvertis yra gaunamas analizuojant struktūrinius grafo elementus (viršūnes, lankus ar pografius). Šios grupės grafai bus nagrinėjami.

Kuo sudėtingesni yra duomenys, tuo didesnę atsakomybę turi analitikas, kada apibrėžia kas yra normalu. Pavyzdžiui, jeigu turime kintantį grafą, pašalinė viršūnė gali būti tokia, kuri turi neįprastai didelį laipsnį, neįprastą ryšį, ar neįprastai kintančius atributus. Iš esmės yra neribotas kiekis būdų kaip įvertinti pašalinius elementus. Net specifiniame originalumo tipe, įverčio modelis gali būti pagrįstas jo laipsnio, gretimumo ar lanko svorio distribucija. Pats originalumo apibrėžimas labai sudėtingam duomenų modeliui, šiuo atveju - grafui, veda į nesuskaičiuojamą kiekį variacijų. Tokiuose scenarijuose svarbu originalumą apibrėžti remiantis sprendžiama užduotimi, nes nėra universalaus originalumo apibrėžimo ir specializuotas apibrėžimas geriau tiktų konkrečiai užduočiai.

4. Originalumo aptikimas viename dideliame grafe

Dideliuose grafuose, neįprastos struktūrinės charakteristikos gali būti panaudojamas originalumui apibrėžti skirtinguose grafo regionuose. Tarkime, kad grafas yra apibrėžiamas taip: $G = (N,A)$, kur N yra viršūnių aibė ir A yra gretimumo matrica. Viršūnių kiekis aibėje N yra n ir lankų kiekis matricoje A yra m, todėl A yra $n \times n$ matrica, kurioje yra $m \leq n^2$ nenulinių įrašų. Daugumoje atvejų, ši matrica yra reta, todėl turime $m \ll n$. Taip pat, jei grafas G yra neorientuotas, matrica yra simetrinė. Pavyzdžiui, jeigu turime tokį grafą kaip 4 pav., jo gretimumo matrica bus:

$$\begin{pmatrix} - & 1 & 1 & 0 \\ 1 & - & 1 & 0 \\ 1 & 1 & - & 1 \\ 0 & 0 & 1 & - \end{pmatrix}$$



4 pav. Neorientuoto grafo pavyzdys

4.1. Viršūnių originalumo aptikimas

Viršūnių originalumas gali būti apibrėžiamas skirtingais būdais. Apibrėžimo esmė išgauti atributus iš viršūnės gretimumo ir tada apibrėžti originalumą tų atributų kontekste. Keletas tokių atributų yra aprašomi darbe [AMF10], kurie gaunami iš vieno žingsnio dydžio gretimumo kaimynystės (pografis, kurį sudaro pasirinkta viršūnė, visos jai gretimos viršūnės ir jas jungiantys lankai), kur pasirinkta viršūnė yra i:

- (Viršūnės atributas n_i). Viršūnės laipsnis (viršūnei i kaimyninių viršūnių kiekis).
- (Lankinis atributas e_i). Lankinis atributas yra lankų kiekis šioje viršūnės i kaimynystėje.
- (Svorių atributas w_i). Svorių atributas yra pritaikomas tik svoriniams grafams, jis atitinką visų lankų svorių viršūnės i kaimynystėje sumą.

Kombinuojant šiuos originalumo atributus, galima gauti įvairių išvestinių atributų:

- Viršūnės ir lankinis atributas: Labai tankios viršūnės kaimynystės, kurios dažnai sudaro grafo kliką (kada visos viršūnės yra gretimos visoms kitoms), o labai retos kaimynystės primena žvaigždės struktūrą. Bendru atveju, proporciją tarp lankų ir viršūnių, kurią yra tikimasi gauti, yra $e_i \propto n_i^{\alpha}, \alpha \in (1,2)$. Išanalizavus grafą, galima surasti idealią proporciją, o proporcingus neatitikimus traktuoti kaip originalumo požymius.

- Svorių ir lankinis atributas: Jeigu aplink viršūnę yra daug lankų, tai lems auštą svorių atributo reikšmę. Ši proporcija, kurią yra tikimasi gauti, yra: $w_i \propto e_i^{\beta}\ \beta \geq 1$.

Taip pat verta paminėti, kad daugelis atributų gali būti išgauti iš grafo kaimynystės, kurie skirtingai apibrėžia originalumo įvertį. Prieš tai paminėti atributai yra tik pagrindiniai, nes jų gali būti žymiai daugiau ir įvairesnių. Ypač svarbu, kad pasirinktas originalumo atributas būtų aktualus sprendžiamai užduočiai, todėl dažnai tokio atributo pasirinkimas reikalauja gilesnio užduoties supratimo.

4.2. Pografių originalumas

Originalus pografis yra apibrėžiamas kaip pagrindinio grafo dalis, kuri pasižymi neįprastu išsidėstymu, lyginant su normaliais grafo išsidėstymais. Paprastai originalūs pografiai negali būti prasmingai apibrėžiami dideliame grafe, nebent grafo viršūnių identifikatoriai (su viršūne susieta informacija) kartojasi. Pavyzdžiui Internete ar socialiniame tinkle, kur kiekviena viršūnė yra unikali, yra žymiai sunkiau suprasti kokie ryšiai yra normalūs. Alternatyviai, jeigu viršūnės neturi jokios indentifikacinės prasmės, tada originalumas nustatomas vien iš struktūrinio panašumo. Toks atvejis yra nagrinėjamas [NC03] darbe.

Kitas originalumo nustatymas, pristatytas [NC03], yra paremtas minimaliu aprašymo ilgio (angl. *minimum description length*) principu. Minimalus aprašo ilgis (funkcija DL) matuoja grafui užrašyti naudojamos informacijos kiekį. Paprasčiausiu atveju tai gali būti lankų kiekis. SUBDUE [CH00] sistemoje, dažnai pasitaikantis reguliarumas yra panaudojamas kaip minimalus aprašymo ilgio etalonas. Pavyzdžiui, jeigu S yra vidinė struktūra, kuri pastoviai pasikartoja pagrindiniame grafe G, tai sutraukus tą vidinę struktūrą iki vienos viršūnės gauname suspaustą grafą, kuris atitinka mažesnį aprašymo ilgį ir gali būti aprašomas kaip suspaustas grafas ir mažesnis pografis. Dažnai pasitaikantis pografis S grafe G leidžia glaustai aprašyti suspaustą grafą aprašymo ilgio $DL(G|S)$ ir dažnai pasikartojančios struktūros S terminais. Glaustumas (mažesnis aprašo ilgis) yra užtikrinamas, nes S turi būti aprašomas tik vieną kartą. Tada aprašo ilgis apibrėžiamas taip:

$$F1(S,G) = DL(G|S) + DL(S) \tag{2}$$

Pografiai, kurie pasitaiko grafe gana dažnai, gaus žemas $F1(S,G)$ reikšmes, nes jie nėra originalūs, neįprasti. Tada, natūrali išvada kyla, kad jeigu $F1(S,G)$ reikšmės yra aukštos, pografis yra automatiškai originalus, tačiau tai nėra visiškai teisinga. Toks originalumo aptikimas nelabai gerai veikia, nes nediskriminuoja pografių pagal jų dydį. Pavyzdžiui, pografiai, kurie turi tik vieną viršūnę, dažnai įgis aukštą originalumo įvertį pagal 2 formulę. Dėl to, reikalinga tam tikra euristika, kuri atsižvelgia ir į pografio dydį. Viena iš jų buvo pristatyta [DAF+10], kuri pasitelkia pagrindine aprašymo ilgio charakteristika (reguliarumo dažnumas) ir ją papildo grafo dydžiu:

$$F2(S,G) = Size(S) \cdot Instances(S,G) \tag{3}$$

Čia, $Size(S)$ atitinka S pografio viršūnių kiekį, o $Instances(S,G)$ atitinka kiek kartų pografis

S yra aptinkamas grafe G. Tada jau galime teigti, kad pografiai, kurie įgija žemą $F2(S,G)$ reikšmę, yra neįprasti.

Kitas metodas yra tiesiogiai paremtas SUBDUE algoritmu. SUBDUE metodas buvo sukurtas iteratyviu būdu aptikti populiarius (dažnai pasikartojančius) pografius. Šis metodas suspaudžia populiarius pografius ir pakeičia juos naujomis viršūnėmis. Populiaresni pografiai yra pakeičiami ankstesnėse iteracijose. Čia ir paaiškėja pagrindinė SUBDUE metodo idėja - jei pografis yra sutraukiamas į viršūnę vėlesnėse iteracijose, vadinasi šis pografis yra retesnis (todėl ir originalesnis), lyginant su tais, kurie buvo sutraukti ankstesnėse iteracijose. Taigi, pografio originalumo įvertis yra aprašomas taip:

$$O = 1 - \sum_{i=1}^{n} \frac{(n-i+1)}{n} \cdot \frac{DL_{i-1}(S) - DL_i(S)}{DL_0(S)} \tag{4}$$

Čia $O \in (0,1)$ yra originalumo matas, o i yra SUBDUE iteracijos indeksas. Aukštos reikšmės reiškia, didesnį originalumą. Aprašo ilgis $DL_i(S)$ bendru atveju mažėja, didėjant iteracijos indeksui i. $\frac{DL_{i-1}(S)-DL_i(S)}{DL_0(S)}$ atitinka suspaudimo dalį i-osios iteracijos metu, o $\frac{(n-i+1)}{n}$ suteikia didesnę reikšmę, jeigu suspaudimas nutinka ankstesnėje iteracijoje. Taip pat, pografio S viršūnių suspaudimo greitis (iteracijos numeris), turės didžiulę įtaką originalumo įverčiui, nes nemažai pografių turės panašius reguliarumus.

4.3. Originalumo nustatymas kintančiuose grafuose

Tokiais atvejais, kada grafo pokyčiai neįprasti ir staigūs, reikėtų gebėti juos atpažinti. Anomalijų atpažinimas ir pokyčių analizė yra glaustai susijusios sritys [FP+99; TY06], ypač kada pokyčiai yra staigūs ir atitinka neįprastus įvykius. Grafų kontekste tokias anomalijas galima kategorizuoti:

- Viršūnių anomalijos: Staigūs pokyčiai viršūnės kaimynystėje.
- Kaimynystės evoliucija: Neįprasti ir dideli kaimynystės pokyčiai yra pažymimi kaip anomalijos [AY05; GAH+11].
- Atstumo evoliucija: Viršūnių poros su neįprastai dideliais lanko svorio pokyčiais yra aptinkamos ir pažymimos kaip anomalijos [GAH11].

4.3.1. Anomalijų aptikimas kaimynysčių evoliucijos metu

Evoliuciniai (atsirandantys evoliucijos eigoje, pastovūs) pokyčiai grafe dažnai sukelia didelių pokyčių grafą sudarančių kaimynysčių struktūroje. Kaimynysčių evoliucijos analizė atskleidžia svarbių įžvalgų apie staigius viršūnių narystės pokyčius, klasterių atsiradimą ir pradingimą, viršūnių narystės klasteriui nepastovumą ar bendrą klasterių kokybės pokytį. Dauguma tokių algoritmų apjungia klasterizavimo palaikymą su evoliucine analize.

4.3.1.1. Klasterių palaikymas ir evoliucinė analizė

Evoliucinės analizės ir klasterizavimo proceso apjungimas yra svarbus siekiant rasti aktualius pokyčių taškus duomenyse. ENetClus metodas [GAH+11] apibendrina NetClus [SYH09] modelį kintančiam atvejui. Tai yra negriežtas klasterizavimo modelis, kuris kiekvienai viršūnei priskiria narystės klasteriui tikimybę. Pagrindinė šio modelio idėja yra atlikti klasterizavimą su duomenų tarpinėmis stadijomis (angl. *snapshot*). Kiekvienoje tarpinėje stadijoje kiekvienos viršūnės tikimybė būti priskirtai tam tikram klasteriui yra apskaičiuojama naudojant NetClus algoritmą. Galutinė tikimybė konkrečioje tarpinėje stadijoje yra naudojama kaip inicializacija sekančiai iteracijai. Tokiu būdu yra užtikrinamas klasterių tęstinumas. Klasteriai naujoje stadijoje gali būti tiesiogiai palyginami su ankstesnėje stadijoje buvusiais. Taip pat, yra siūlomi keli evoliucijos atributai, kurie skirti matuoti klasterių charakteristikas ir jų kaitą. Tokių atributų smarkus pokytis gali būti traktuojamas kaip anomalija grafe. Keletas tokių atributų pavyzdžių:

- Klasterio narystės vientisumas: Viršūnės narystės klasteriams tikimybių vektorius, kuris yra gaunamas negriežtuose klasterizavimo algoritmuose kaip ENetClus, yra naudojamas kartu su prieš tai buvusios stadijos tikimybių vektoriumi apskaičiuojant kosinuso panašumą (angl. *cosine similarity*), kas gali būti laikoma kaip pokytis, nuo kurio dydžio priklauso ir anomalijos požymis.

$$cossim(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

- Klasterių tarpinių stadijų kokybė: Tarpklasterinio panašumo ir vidinio klasterių panašumo santykis yra panaudojamas kaip kokybės įvertis. Dideli pokyčiai klasterių kokybėje iš eilės einančių tarpinėse stadijose reiškia didelius esminius grafo klasterizavimo pokyčius, kas atitinka anomaliją.

- Klasterių naujovės, suliejimai, atskyrimai ir išnykimai: Naujų klasterių formavimas žymi kažką naujo duomenyse. Akivaizdu, kad įvykus klasterių suliejimui, atskyrimui ar išnykimui yra tikėtini dideli struktūriniai pokyčiai. Kiekvienas iš tokių pokyčių yra aprašomas [GAH+11].

- Objektų stabilumas: Jeigu klasteris, kuriam priklauso objektas, pastoviai kinta, akivaizdu, kad toks objektas iš principo negali būti stabilus ir traktuojamas kaip anomalija. Objekto stabilumas yra apibrėžiamas kaip iš eilės einančių tarpinių būsenų dalis, kurioje objektas nekeičia klasterio narystės. Tada anomalijos įvertis yra atvirkščiai proporcingas objekto stabilumui.

- Objektų socialumas: Objektai, kurie priklauso keliems skirtingiems klasteriams yra laikomi socialiais. Tai reiškia, kad narystės tikimybės vektoriaus reikšmės yra pasiskirsčiusios po kelis klasterius, kai yra naudojamas negriežto klasterizavimo (apskaičiuoja viršūnės narystės klasteriui tikimybes, o ne priskiria vieną konkretų klasterį) algoritmas . Socialumas

skiriasi nuo viršūnės laipsnio, kadangi jis yra matuojamas pagal skirtingų kaimynysčių elgesį (kaip kita objekto ryšis su tam tikromis kaimynystėmis). Tarkime, $p_1 ... p_k$ yra objekto narystės tikimybės k skirtingiems klasteriams. Tokiu atveju Gini koeficientas yra gaunamas $1 - \sum_{i=1}^{k} p_i^2$, o entropija $\sum_{i=1}^{k} p_i \cdot \frac{ln(p_i)}{k}$. Tada galima atlikti matuojant klasterių narystės tikimybes naudojant Gini koeficientą arba objekto entropiją. Aukštesnės reikšmės reiškia didesnį socialumą. Pavyzdžiui, bibliografiniame grafe, tai galėtų reikšti autorių bendradarbiavimą tarp daugybės skirtingų sričių. Staigūs socialumo pokyčiai gali atitikti staigius pokyčius bendradarbiavimo elgesyje.

4.3.1.2. Kaimynystės analizė

Smarkiai mažėjančios ar didėjančios kaimynystės gali būti aptinkamos su diferencialinio grafo pagalba (angl. *differential graph*) [AY05]. Mažėjančios ar didėjančios kaimynystės priklauso nuo lankų ir viršūnių kiekio pokyčio toje kaimynystėje. Tarkime, turime lanko (i,j) svorį $w_{ij}(t)$ laiko momente t. Siekiant sukonstruoti diferencialinį grafą (žymima $\Delta G(t_1,t_2)$), reikia sukonstruoti normalizuotą grafą laiko momentais t_1,t_2. Normalizuotas grafas $G(t) = (N(t),A(t))$ laiko momente t, kur $N(t)$ yra viršūnių aibė laiko momente t, o $A(t)$ yra gretimumo matrica laiko momente t, žymimas $\overline{G(t)}$. $\overline{G(t)}$ turi lygiai tokias pačias viršūnes ir lankus kaip ir $G(t)$, tačiau lankų svoriai gali skirtis. Tada $W(t) = \sum_{(i,j)\in A} w_{ij}(t)$ yra visų grafo $G(t)$ lankų svorių suma ir normalizuotas svoris $\overline{w_{ij}(t)}$ yra gaunamas iš $\frac{w_{ij}}{W(t)}$. Tada diferencialinis grafas gali būti sukonstruojamas redukuojančio proceso pagalba remiantis tarpinėmis būsenomis laiko momentuose t_1,t_2. Dėl to diferencialinis grafas $\Delta G(t_1,t_2)$ turės tas pačias viršūnes ir lankus, kaip ir $G(t_2)$, o jo diferencialinis svoris yra aprašomas $\Delta w_{ij}(t_1,t_2) = \overline{w_{ij}(t_2)} - \overline{w_{ij}(t_2)}$. Tokiu atveju, jeigu lankas (i,j) neegzistuoja grafe $\overline{G(t_1)}$, tada $w_{ij}(t_1)$ reikšmė prilyginama nuliui. Normalizavimo metu diferencialiniai svoriai gali būti neigiami, kas rodo, kad interakcija (bet koks struktūros pokytis: svorio, viršūnės, lanko ir lanko krypties) šalia tų lankų žymiai sumažėjo evoliucijos metu. Pavyzdžiui, bibliografiniame grafe, jeigu sumažėja autorių poros publikacijų leidimo tempas, atitinkamų lankų svoriai diferencialiniame grafe bus neigiami. Turint diferencialinį grafą, dažnai yra norima nustatyti smarkiai kintančius pografius. Lankai, kurie turi aukštus teigimus ar neigiamus diferencialinius svorius dažnai atitinka evoliucionuojančius (kintančius) objektus. Siekiant identifikuoti mažėjančias ar didėjančias kaimynystes, reikia nustatyti kurie pografiai pasižymi vien tik aukštomis teigiamomis ar neigiamomis interakcijomis. Tai yra žymiai sudėtingesnė užduotis negu klasterių radimas $\Delta G(t_1,t_2)$ grafe. Metodai tokiai užduočiai spręsti yra aptariami [AY05] darbe.

4.3.2. Trumpiausio kelio atstumu paremtas anomalijų aptikimas

Dauguma realaus pasaulio grafų, kaip Internetas ar socialiniai tinklai, pastoviai kinta kelio tarp viršūnių porų atžvilgiu. Pavyzdžiui, buvo parodyta, kad dauguma realaus pasaulio grafų pasižymi mažėjančiais diametrais (didžiausia reikšmė iš visų viršūnių porų trumpiausių atstumų, kitaip tariant, labiausiai nutolusių viršūnių trumpiausias kelias) [BHK+06]. Taip yra todėl, kad nuo pastovaus lankų pridėjimo jie tankėja. Remiantis tokia prielaida, staigūs ir netikėti pokyčiai rodo neįprastus grafo įvykius. Pavyzdžiui, bibliografiniame grafe, dauguma viršūnių ir konkrečiame

regione yra apjungtos mažais (2 ar 3 žingsnių) keliais, todėl viršūnių apjungimas, kurių minimalus kelias prieš tai buvo iš 5 žingsnių, yra laikomas neįprastu įvykiu. Tas įvykis tikriausiai reiškia dviejų skirtingų sričių autorių bendradarbiavimą. Paprastas tokios anomalijos apskaičiavimas yra pasiekiamas randant visų įmanomų porų trumpiausią kelią dviejose skirtingose tarpinėse stadijose (t_1,t_2). Tokiu atveju anomalijos bus aptinkamos kaip viršūnių poros, kurių trumpiausio kelio įvertis pasikeitė dramatiškai. Akivaizdu, kad toks metodas reikalauja daug atminties, ypač jeigu dirbama su dideliais grafais. Pavyzdžiui, jeigu grafas turi 10^8 viršūnių, tai tokių įmanomų porų yra 10^{16}, kas tiesiog nėra patogu ar praktiška. Dėl to yra svarbu sukurti tokius metodus, kurie galėtų efektyviai identifikuoti k didžiausius trumpiausio kelio pokyčius euristiniu būdu. Verta paminėti, kad svarbūs lankai yra tokie, kurie aptinkami daugelyje trumpiausių kelių, nepriklausomai nuo analizuojamos tarpinės būsenos. Randomizuotas euristinis algoritmas yra aprašomas [GAH11] būtent tokių svarbių lankų radimui. Tada jie yra panaudojami svarbių viršūnių porų identifikavimui, kur galimai įvyko smarkus pokytis.

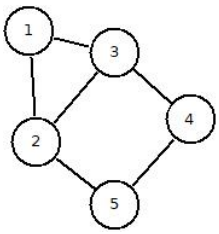
Lankų svarbumo algoritmas aprašomas taip:

Įvedimas: Grafas $G(V,E)$, su n viršūnių ir m lankų
Išvedimas: Vidutinis kiekvieno lanko $e \in E$ svarbumas

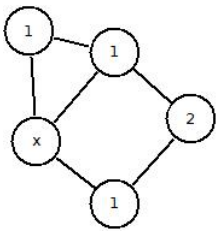
1: Atsitiktinai atrinkti α viršūnių iš grafo G į aibė S
2: Inicializuoti svarbumo įvertį $I(e)$ nuline reikšme kiekvienam lankui $e \in E$
3: **for** kiekvienai viršūnei $x \in S$ **do**
4: Apskaičiuoti trumpiausius atstumus nuo pradinės viršūnės x (pavyzdžiui naudojant Djikstros algoritmą), gauti trumpiausių kelių medį SPT_x
5: Peržymėti SPT_x viršūnes gautomis atstumų reikšmėmis
6: Kiekvienam lankui $(i,j) \in E$, nustatyti, ar šis lankas yra įtemptas (paaiškinimas žemiau) SPT_x medyje, ir rasti visus jam alternatyvius lankus
7: Pasirinkti β atsitiktinių trumpiausio kelio medžių (žymimų SPT_{xy}), kurie gaunami pakeičiant pradinio SPT_x medžio γ įtemptų lankų jiems alternatyviais (alternatyvūs trumpiausių atstumų medžių pavyzdys yra pavaizduotas pav. 7)
8: **for** Kiekvienam SPT_{xy} trumpiausio kelio medžiui **do**
9: Apytiksliai apskaičiuoti svarbumo įvertį $I(e, SPT_{xy})$ kiekvienam lankui e, SPT_{xy} medžio kontekste, kaip lanko vaikų (visi kiti lankai, kurie SPT_{xy} medyje gali būti pasiekiami nuo x viršūnės einant per tėvinį lanką) skaičių.
10: Pridėti gautą svarbumo įvertį $I(e, SPT_{xy})$ prie $I(e)$
11: **end for**
12: **end for**
13: Apskaičiuoti vidutinį kiekvieno lanko svarbumą normalizuojant $I(e)$ su β
14: **return** Vidutinis kiekvieno lanko $e \in E$ svarbumo įvertis

---

Atstumo žyma $d_x(i)$ viršūnei i trumpiausio kelio medyje SPT_x, yra apibrėžiama kaip trumpiausio kelio atstumas nuo x iki i viršūnės (pvz. kaip parodyta pav. 6). Tada galima formalizuoti įtempto lanko apibrėžimą. Lankas (i,j) yra įtemptas trumpiausių atstumų medyje SPT_x, jeigu $d_x(j) = d_x(i) + w(i,j)$ arba $d_x(i) = d_x(j) + w(i,j)$, kur $w(i,j)$ yra lanko svoris. α, β ir γ yra eksperimentiniu būdu parenkamos konstantos. Lanko svarbumo įvertis (grafe $G(N,E)$) $I(e), e \in E$ yra apytiksliai lygus tikimybei, kad lankas e bus atsitiktinai parinktame trumpiausių kelių medžio grafe.

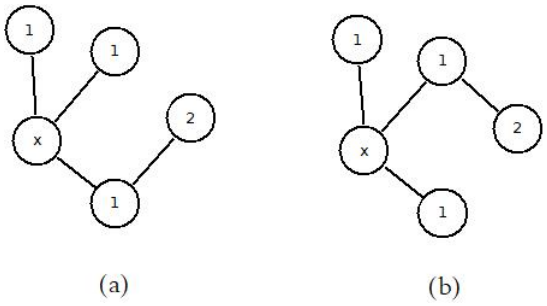


5 pav. Paprastas grafas



6 pav. Atstumo žymių grafas, pagal gautus atstumus nuo pradinės viršūnės x iš grafo, pavaizduoto pav. 5

.



7 pav. Trumpiausių atstumų žymių medžių alternatyvos nuo pradinės viršūnės x, gaunamos iš grafo, pavaizduoto pav. 6

4.4. Originalumo aptikimo grafuose santrauka

Originalumo aptikimas grafuose apibrėžiamas kaip neįprastumų ar anomalijų aptikimas. Toks požymis gali būti gaunamas skirtingais būdais, kurie nevienodai tinkai visiems grafams. Kadangi grafų gali būti sudėtingų ir labai įvairių, todėl ir metodų yra daug ir įvairių, kuriuose normalumas yra apibrėžiamas skirtingai. Originalumo požymis gali būti apibrėžiamas naudojant viršūnių, lankų, pografių, kintančių pografių ar kintančių atstumų informaciją. Dauguma grafo anomalijų aptikimo algoritmų analizuoja kaimynystės pokyčius ar jų išvestinę formą. Originalumo aptikimas remiantis anomalijomis yra įmanomas, tačiau anomalijos dažnai reikalauja sprendinių aibės monotoniškumo. Tai yra, jeigu visi (ar dauguma) sprendinių yra žymima kaip anomalijos, iš to naudos nėra, nes tos pačios anomalijos nublanksta. Geriausiu atveju anomalijos sudaro mažą sprendinių proporciją.

5. Tradiciniai euristiniai algoritmai grafų optimizavimo uždaviniams

Siekiant įvertinti originalumo euristikos efektyvumą, jis bus lyginamas su keliais tradiciniais euristiniais algoritmais, kurie matuoja progresą tikslo funkcijos pagalba. Sprendžiami grafų optimizavimo uždaviniai yra keliaujančio pirklio (angl. *Traveling Salesman Problem*) (toliau TSP) ir grafų spalvinimo (angl. *Graph Coloring Problem*) (toliau GCP). Bendru atveju GCP aprėpia viršūnių arba lankų spalvinimą, šiuo atveju bus viršūnių spalvinimas.

Grafas, kuriame vyks optimizavimas uždavinių sprendimas, yra apibrėžiamas taip: $G = (N,A)$, kur N yra viršūnių aibė ir A yra gretimumo matrica. Viršūnių kiekis aibėje N yra n ir lankų kiekis matricoje A yra m, todėl A yra $n \times n$ matrica, kurioje yra $m \leq n^2$ nenulinių įrašų. Taip pat, grafas G yra neorientuotas ir svorinis.

Keliaujančio pirklio uždavinys reikalauja rasti trumpiausią ciklą (kelią) tarp visų viršūnių, kuris prasideda ir baigiasi toje pačioje viršūnėje. Grafų spalvinimo uždavinys reikalauja rasti tokias spalvas viršūnėms, kad bet kurios dvi gretimos viršūnės nebūtų tos pačios spalvos. Be to, pageidautina, kad panaudotų spalvų kiekis būtų kuo mažesnis (optimaliu atveju mažiausias). GCP atveju, grafo lankų svoriai nėra aktualūs. Abu uždaviniai yra NP (angl. *nondeterministic polynomial time*) klasės uždaviniai, todėl pasižymi apgaulingumu.

5.1. Atkaitinimo modeliavimas

Atkaitinimo modeliavimo [Bro11] (angl. *simulated annealing*) algoritmas yra modeliuojamas pagal panašią veiklą metalurgijoje. Atkaitinimo metu, metalas yra pakankamai įkaitintas, kad jo molekulinė struktūra galėtų kisti. Tada temperatūra lėtai mažinama, kas sumažina energijos kiekį atomuose, todėl metalo forma galiausiai sukietėja ir nekinta. Tokio pobūdžio algoritmas bendru atveju aprašomas taip:

Įvedimas: $ProblemSize, iterations_{max}, temp_{max}$
Išvedimas: S_{best}

1: $S_{current} \leftarrow CreateInitialSolution(ProblemSize)$
2: $S_{best} \leftarrow S_{current}$
3: **for** i = 1 to $iterations_{max}$ **do**
4: $\quad S_i \leftarrow CreateNeighborSolution(S_{current})$
5: $\quad temp_{curr} \leftarrow CalculateTemperature(i, temp_{max})$
6: $\quad$ **if** Cost(S_i) $\leq Cost(S_{current})$ **then**
7: $\quad\quad S_{current} \leftarrow S_i$
8: $\quad\quad$ **if** Cost(S_i) $\leq Cost(S_{best})$ **then**
9: $\quad\quad\quad S_{best} \leftarrow S_i$
10: $\quad\quad$ **end if**
11: $\quad$ **else if** $Exp(\frac{Cost(S_{current}) - Cost(S_i)}{temp_{curr}}) > Rand()$ **then**
12: $\quad\quad S_{current} \leftarrow S_i$
13: $\quad$ **end if**
14: **end for**
15: **return** S_{best}

Čia $ProblemSize$ yra sprendinių aibės dydis, $iterations_{max}$ yra iteracijų kiekis, o $temp_{max}$ yra pradinė temperatūra. Funkcija $CreateInitialSolution$ sukuria pradinių atsitiktinių sprendinių aibę, o funkcija $CreateNeighborSolution$ sukuria kaimyninių (kurie nedaug skiriasi, pvz. vienu lanku, ar viena viršūnių pora) sprendinių aibę. $CalculateTemperature$ funkcija aprašo temperatūrą (pvz. geometrinė nykstamoji progresija) iteracijoje i. $Rand$ funkcija gauna atsitiktinę reikšmę intervale $(0,1)$. Funkcija $Cost$ naudojama apskaičiuoti sprendinio kainą, kuri yra minimizuojama.

Algoritmo pagrindinė dalis abiejų uždavinių (TSP ir GCP) atveju nekinta, tačiau kinta funkcijos $CreateInitialSolution$, $CreateNeighborSolution$ ir $Cost$.

5.2. Skruzdžių kolonijos sistema

Skruzdžių kolonijos sistema [Bro11] (angl. *ant colony system*) yra modeliuojama pagal panašią skruzdžių veiklą gamtoje ieškant maisto. Po daugybės kelionių nuo skruzdėlyno iki maisto, takuose yra paliekami feromonai (angl. *pheromones*). Šiais feremonais yra pažymimi dažnai naudojami keliai, kurie palaipsniui išgaruoja, todėl senesni keliai turės mažiau feremonų nei naujesni. Taip pat populiaresni keliai turės daugiau feromonų nei mažiau naudojami. Feromonai daro įtaką kelio pasirinkimui - kuo daugiau feromomų, tuo kelias yra labiau tikėtinas pasirinkime. Tokio pobūdžio algoritmas bendru atveju aprašomas taip:

Įvedimas: $ProblemSize, Population_{size}$,$iterations_{max}, m, \rho$,$\beta$,$\gamma$,$\theta$
Išvedimas: S_{best}

1: $S_{best} \leftarrow CreateHeuristicSolution(ProblemSize)$
2: $Best_{cost} \leftarrow Cost(S_{best})$
3: $Pheromone_{init} \leftarrow \frac{1.0}{ProblemSize \times Best_{cost}}$
4: $Pheromone \leftarrow InitializePheromone(Pheromone_{init})$
5: **for** j = 1 to $iterations_{max}$ **do**
6: **for** i = 1 to m **do**
7: $S_i \leftarrow ContructSolution(Pheromone,ProblemSize,\beta,\theta)$
8: $Si_{cost} \leftarrow Cost(S_i)$
9: **if** $Si_{cost} \leq Best_{cost}$ **then**
10: $Best_{cost} \leftarrow Si_{cost}$
11: $S_{best} \leftarrow S_i$
12: **end if**
13: $UpdateAndDecayPheromone(Pheromone, S_i, Si_{cost}, \gamma)$
14: **end for**
15: $UpdateAndDecayPheromone(Pheromone, S_{best}, Best_{cost}, \rho)$
16: **end for**
17: **return** S_{best}

Čia θ yra lokalios feromonų istorijos koeficientas, kuris nustato feromonų įtaką sprendinio komponentų (pvz. TSP atveju - viršūnių pagal lankų feromonus) pasirinkime. β yra koeficientas, kuris nustato kokią įtaką turi godi euristika (pvz. TSP atveju - rinktis viršūnę į kurią lanko svoris mažiausias). θ ir β koeficientai apskaičiuoja sprendinio komponento pasirinkimo tikimybę. ρ ir γ yra feromonų garavimo koeficientai, kurie nustato kaip greitai nyksta feromonai (atitinkamai

po vienos skruzdės iteracijos ir po visų). θ yra godumo koeficientas, kuris nusako kaip dažnai renkantis sprendinio komponentą ignoruoti feromonus ir naudoti godžią strategiją. Galiausiai, m yra skruzdžių kiekis. Funkcija $CreateHeuristicSolution$ sukuria pradinį sprendinį remiantis kokia nors godžia euristika (pvz. TSP atveju arčiausių kaimynų, kada pasirenkami artimiausios kaimyninės viršūnės). Funkcija $InitializePheromone$ inicializuoja feromonus, pagal tai, koks buvo pradinis godžios euristikos atstumas. Funkcija $ContructSolution$ sukonstruoja sprendinį renkantis jo komponentus pagal tikimybes, kurios gaunamos iš feromonų ir godžios euristikos koeficientų. Funkcija $LocalUpdateAndDecayPheromone$ atnaujina feromonų aibę, pagal gauto sprendinio komponentus ir sprendinio kainą. Analogiškai, po visų skruzdžių iteracijos, yra atnaujinami feromonai pagal geriausio (iki šiol rasto) sprendinio komponentus. Funkcija $Cost$ naudojama apskaičiuoti sprendinio kainą, kuri yra minimizuojama.

Algoritmo pagrindinė dalis abiejų uždavinių (TSP ir GCP) atveju nekinta, tačiau kinta funkcijos $CreateHeuristicSolution, InitializePheromone, ContructSolution$, $UpdateAndDecayPheromone$ ir žinoma $Cost$.

5.3. Genetinis algoritmas

Genetinis algoritmas [Bro11] (angl. *genetic algorithm*) modeliuoja evoliuciją, kuri perneša genetinius bruožus. Nauja generacija yra sukuriama poruojant tėvinius individus kryžminimo (individų poros genai yra kombinuojami į naują individą, kuris turi abiejų tėvų genų dalis) būdu ir atsitiktinėmis kopijavimo klaidomis, vadinamomis mutacijomis. Geriau aplinkai prisitaikę individai turės daugiau įtakos sekančiai generacijai. Prasti individai bus pašalinami ir pakeičiami naujais. Tokio pobūdžio algoritmas bendru atveju aprašomas taip:

Įvedimas: $ProblemSize, Population_{size}, P_{crossover}, P_{mutation}, iterations_{max}$
Išvedimas: S_{best}

1: $Population \leftarrow InitializePopulation(Problem_{size})$
2: $EvaluatePopulation(Population)$
3: $S_{best} \leftarrow GetBestSolution(Population)$
4: **for** i = 1 to $iterations_{max}$ **do**
5: $\quad Parents \leftarrow SelectParents(Population, Population_{size})$
6: $\quad Children \leftarrow \emptyset$
7: $\quad$ **for** $Parent_1, Parent_2 \in Parents$ **do**
8: $\quad\quad Child_1, Child_2 \leftarrow Crossover(Parents_1, Parent_2, P_{crossover})$
9: $\quad\quad Children \leftarrow Mutate(Child_1, P_{mutation})$
10: $\quad\quad Children \leftarrow Mutate(Child_2, P_{mutation})$
11: $\quad$ **end for**
12: $\quad EvaluatePopulation(Children)$
13: $\quad S_{best} \leftarrow GetBestSolution(Children)$
14: $\quad Population \leftarrow Replace(Population, Children)$
15: **end for**
16: **return** S_{best}

Čia $Population_{size}$ yra populiacijos dydis (kiek vienu metu gali būti individų), $P_{crossover}$ yra kryžminimo tikimybė, o $P_{mutation}$ yra mutacijos tikimybė. Funkcija $InitializePopulation$ ini-

cializuoja pradinę populiaciją, o $EvaluatePopulation$ ją įvertina (apskaičiuoja kiekvieno sprendinio kainą, kuri priklauso nuo uždavinio). Funkcija $GetBestSolution$, gauna geriausią individą dabartinėje populiacijoje. Funkcija $SelectParents$ išsirenka tėvus, kurie bus naudojami naujai generacijai kurti. $Crossover$ ir $Mutation$ funkcijos atitinkamai atlieka kryžminimą ir mutaciją (pagal tikimybes $P_{crossover}$ ir $P_{mutation}$). Galiausiai, $Replace$ funkcija pakeičia blogiausius populiacijos individus naujais.

Algoritmo pagrindinė dalis abiejų uždavinių (TSP ir GCP) atveju nekinta, tačiau kinta funkcijos $InitializePopulation, EvaluatePopulation, Crossover, Mutate$. Genetinė informacija yra dažniausiai sutapatinama su sprendinio komponentais (pvz. TSP atveju, genai gali būti viršūnės, kai visas individas atitinka kelią). Šią genetinę informaciją $Crossover$ funkcija išardo ir suklijuoja kitaip, todėl yra gaunamos abiejų tėvinių sprendinių dalys viename.

6. Originalumo paieška ir jos variacijos

6.1. Originalumo paieškos idėjos konkretizavimas

Nors, originalumo paieškos (*Novelty Search*) (toliau NS) metodiką teoriškai galima pritaikyti bet kokiam euristiniam algoritmui, kuris remiasi tikslo funkcija, ji buvo sukurta genetiniam algoritmui, kuriuo šis darbas ir apsiribos. Pagal NS autoriaus [LS08b] yra atskiriamos sprendinio ir sprendinio elgesio idėjos. Pavyzdžiui, roboto navigacijos simuliacijoje sprendinys yra roboto konfigūracija (pvz. dirbtinis neuroninis tinklas), o roboto elgesys yra navigacijos simuliacijos galutinis taškas. Vienas iš pagrindinių NS aspektų yra sprendinių **elgesių** archyvas ir to archyvo valdymas.

Dažniausiai šis archyvas yra įprastas dinaminis masyvas, pastoviai plečiamas algoritmo metu. Archyvo plėtimo metu yra svarbūs šie aspektai:

- Kada įtraukti į archyvą sprendinio elgesį. Vienas iš tokių metodų yra panašumo ribos (angl. *threshold*). Šiuo metodo metu yra naudojama panašumo riba, kuri nustato ar sprendinio elgesys pateks į archyvą. Taip pat yra ribojamas elgesių įterpimo kiekis algoritmo iteracijos metu. Tokiu būdu yra išvengiama elgesių dubliavimo ir begalinio archyvo plėtimo. Šis metodas didžiąją dalį atsakomybės perduoda originalaus elgesio atstumo funkcijai. Jeigu ši funkcija per mažai detali, tai archyvas labai greitai užsipildys elgesiais, kurie aprėps visą sprendinių aibę. Tai yra, nebus įmanoma atrasti tokio elgesio, kuris pakankamai skirtųsi nuo esamų. Analogiškai, jeigu elgesio panašumo funkcija yra per daug detali, elgesių aibė bus per daug tiršta ir archyvas per greitai užsipildys. Tą dalinai išsprendžia dinaminė riba, kuri kinta nuo elgesių įtraukimo kiekio šios iteracijos metu ir norimo elgesio įtraukimo dydžio.

- Kaip atpažinti nematytą (originalų) elgesį. Čia ir yra darbo esmė, kas yra aprašoma vėlesniuose skyreliuose.

6.1.1. Originalaus elgesio atstumo funkcija

Svarbiausias NS elementas yra gebėjimas įvertinti dviejų sprendinių elgesių panašumą. NS algoritmo kontekste ši funkcija yra naudojama vietoj tikslo funkcijos. Tai yra, kiekvienas naujo (ar išlikusio seno tam tikrų algoritmų eigoje kaip genetinis algoritmas) sprendinio elgesys įgauna originalumo įvertį šios iteracijos metu. Toks įvertis yra žymiai brangesnis kompiuterio resursų prasme, nes archyvas algoritmo eigoje vis didėja, todėl tai lemia algoritmo sulėtėjimą. Paprastoje NS versijoje, kiekvieno sprendinio elgesys yra matuojamas su kiekvienu esančiu archyve. Toks matavimų kiekis auga tiesiškai, nuo archyvo dydžio, visgi šis kiekis yra žymiai didesnis, lyginant su to paties euristinio algoritmo standartine versija (naudojant tikslo funkciją). Taip pat, efektyvi sprendinio originalumo elgesio atstumo funkcija yra aktuali, nes NS algoritme vis tiek dalyvauja tradicinė tikslo funkcija, kuri neturi įtakos paieškos reguliavimui. Todėl išeina žymiai didesnis funkcijos įverčių kiekis, kas ir yra pagrindinis algoritmo našumo matavimo būdas.

7. Keliaujančio pirklio uždavinio eksperimentų sąlygų konkretizavimas

Keliaujančio pirklio uždavinys (angl. *Traveling Salesman Problem*) (toliau TSP) yra rasti trumpiausią maršrutą pilname grafe, kuris apeina visas viršūnes jų nekartojant ir grįžta į pradinę. Šiame uždavinyje yra naudojami gerai žinomi pilni, simetriški (viršūnės jungia vienas lankas, kelionės kainą į abi puses yra ta pati) tačiau nelabai didelio masto (viršūnių kiekis iki 100) grafai. Yra sprendžiami uždaviniai generuojami ir iš TSPLIB [Rei] duomenų bibliotekos: att48, berlin52, bays29, G(4), G(8), kurių yra žinomas optimalus sprendinys.

7.1. Panašumo funkcijos

Panašumo funkcijos yra naudojamos tik NS algoritme.

7.1.1. Tikslo funkcijos įverčio nuotolis

Primityviausia panašumo funkcija yra dviejų sprendinių kelionės ilgio skirtumas. Tai yra, jeigu dviejų sprendinių kelionės kaina mažai skiriasi, jie yra panašūs. Taip pat, ši funkcija gali būti naudojama kaip diskretizuota (įgauna reikšmes 0 arba 1), kai aktualu ar sprendinio kaina jau buvo matyta ar ne.

7.1.2. Ilgiausia bendra viršūnių seka

Viena iš paprastesnių panašumo funkcijų yra ilgiausios bendros viršūnių sekos. Lyginant du sprendinius, yra randama ilgiausia bendra viršūnių seka. Pavyzdžiui, jeigu turime du individus, jų ilgiausia bendra viršūnių seka bus (2 7 1 6) pagal:

(3 4 8 2 7 1 6 5)

(4 2 7 1 6 8 3 5)

Tada panašumo įvertis yra gaunamas iš tokio santykio:

$$f(L) = \frac{L}{S} \tag{5}$$

Čia L yra ilgiausios sekos ilgis, o S yra sprendinio ilgis (statinis, nes visi sprendiniai vienodo ilgio).

7.1.3. Sutampančių lankų kiekis

Kita visai nesudėtinga panašumo funkcija yra sutampančių lankų kiekio įvertis. Pavyzdžiui, jeigu turime du individus, sutampantys lankai butų (2 7), (7 1), (1 6) ir jei grafas yra simetriškas (3 5) pagal:

(3 4 8 2 7 1 6 5)

(4 2 7 1 6 8 3 5)

Tada panašumo įvertį gauname taip:

$$f(L) = \frac{L}{S} \tag{6}$$

Čia L yra sutampančių lankų kiekis, o S yra sprendinio ilgis (statinis, nes visi sprendiniai vienodo ilgio).

7.2. Sprendinio lokalūs pokyčiai (mutacijos)

TSP sprendinio reprezentacija sutampa visuose nagrinėjamuose algoritmuose. Tai yra viršūnių sąrašas, kuris atitinka kelionę, pavyzdžiui: (1 2 3 4 5 6). Euristinių algoritmų neatskiriama dalis yra gebėjimas „vaikščioti“ po paieškos erdvę, kitaip tariant, sprendinių skaldymas. Tai išreiškiama nedideliais pokyčiais esamam sprendiniui, genetiniame algoritme tai dar yra žinoma kaip mutacijos. Toliau išvardintos mutacijos yra naudojamos šio darbu metu suplanuotuose eksperimentiniuose tyrimuose.

7.2.1. Viršūnės perkėlimas

Atsitiktinai pasirenkama viršūnė ir yra perkeliama į kitą, atsitiktinai parinktą poziciją [MJK92]. Pavyzdžiui čia yra perkeliama 2-oje pozicijoje esanti viršūnė į 4-ą:

$$(1\ 2\ 3\ 4\ 5) \rightarrow (1\ 3\ 4\ 2\ 5)$$

7.2.2. Dviejų viršūnių apkeitimo mutacija

Atsitiktinai yra pasirenkamos dvi skirtingos viršūnės ir sukeičiamos vietomis [Ban90]. Pavyzdžiui čia yra sukeičiamos 2-oje pozicijoje esanti viršūnė su 4-ąja:

$$(1\ 2\ 3\ 4\ 5) \rightarrow (1\ 4\ 3\ 2\ 5)$$

7.2.3. Kelio dalies nukirtimo ir regeneravimo mutacija

Autoriaus siūlomos mutacijos metu yra atsitiktinai pasirenkamas nukirtimo taškas. Tada atsitiktinai pasirenkama kuri dalis bus regeneruota (kairė ar dešinė). Galiausiai nukirsta dalis yra regeneruojama su pasirinkta euristika. Eksperimentų atveju yra naudojama artimiausio kaimyno godžioji euristika. Ji pasirenka dar neturimus kaimynus pagal tai, prie kurio atstumas yra mažiausias.

Pavyzdžiui čia yra atkirtimo pozicija 3 ir regeneruojama iš kairės, renkantis artimiausią neaplankytą viršūnę.

$$(1\ 2\ 3\ 4\ 5) \rightarrow (\mathrm{x}\ \mathrm{x}\ 3\ 4\ 5) \rightarrow (2\ 1\ 3\ 4\ 5\ )$$

7.2.4. Centrinės inversijos mutacija

Šios mutacijos [GH88] metu sprendinys yra padalinamas į dvi sekcijas pagal atsitiktinai pasirinktą poziciją. Tada tos sekcijos yra apverčiamos. Pavyzdžiui čia yra pasirenkama 3 pozicija ir apverčiamos gautos sekcijos:

$$(1\ 2\ 3\ |\ 4\ 5) \rightarrow (3\ 2\ 1\ |\ 5\ 4)$$

7.2.5. Vidinės sekos apvertimo mutacija

Šios mutacijos [GH88] metu yra atsitiktinai pasirenkamos dvi skirtingos pozicijos, kurios sudaro vidinę viršūnių seką, kuri bus apverčiama. Pavyzdžiui čia yra pasirenkama 2 bei 5 pozicija ir apverčiama seka gauta tarp tų pozicijų:

$$(1\ 2\ |\ 3\ 4\ 5\ |\ 6\ 7) \rightarrow (1\ 2\ |\ 5\ 4\ 3\ |\ 6\ 7\ )$$

7.3. Genetinio algoritmo pritaikymas

Genetinis algoritmas (toliau GA), paiešką vykdo dvejomis operacijomis - mutacijos ir kryžminimas. GA metu yra svarbi sprendinio reprezentacija, kurią galima būtų skaldyti (panašiai kaip skaidosi chromosomos organizmuose). Esanti reprezentacija, yra tinkama. Svarbesnė GA dalis yra kryžminimas (angl. *Crossover*). Yra nemažai kryžminimo operacijų pritaikytų TSP, tačiau lyginamos yra šios:

- Daliniai perduodamas kryžminimas (angl. *Partially mapped crossover*) (PMX)
- Eilės numerio kryžminimas (angl. *Order crossover*) (OX)
- Ciklinis kryžminimas (angl. *Cycle crossover*) (CX)

Algoritmo metu yra naudojama viena ar kelios mutacijų ir kryžminimų operacijų kombinacijos. Yra keičiama operacijos tikimybės proporcija, tokiu būdu panaudojant skirtingas operacijas, tačiau nevienodai dažnai. Tokiu būdu yra įmanoma atrasti kombinaciją, kuri naudoja visas operacijas, kas gali būti palankiau nei vieną iš galimų. Tik eksperimentiniu būdu yra galima nustatyti kokia operacijų kombinacija yra palankiausia.

7.3.1. Dalinai perduodamas kryžminimas

Dalinai perduodamo kryžminimo operacija [GL+85] (toliau PMX) perduoda dalį informacijos iš tėvų į vaikus. Dalis vieno iš tėvų genų yra perduodamas vaikui, o likusi informacija yra apsikeičiama tarpusavyje. Pavyzdžiui, tarkime turime du tėvus: (1 2 3 4 5 6 7 8) ir (3 7 5 1 6 8 2 4). PMX operacija suformuoja vaiką pasirinkdama atsitiktinai 2 atkirčio taškus šiuose tėvus atitinkančiuose viršūnių eilutėse. Tarkime, pirmas atkirčio taškas yra po trečio elemento, o antras po šešto. Tada gauname tokią reprezentaciją:

$$\text{Tėvas 1 } (1\ 2\ 3\ |\ 4\ 5\ 6\ |\ 7\ 8)$$

Tėvas 2 (3 7 5 | 1 6 8 | 2 4)

PMX operacija šiame pavyzdyje gauna šiuos dalinio sprendinio atitikimus: $4 \leftrightarrow 1, 5 \leftrightarrow 6$ ir $6 \leftrightarrow 8$. Tada dalinai sugeneruoti vaikai atrodo šitaip, kur x žymi dar neįterptą elementą:

Vaikas 1 (x x x | 1 6 8| x x)

Vaikas 2 (x x x | 4 5 6 | x x)

Tada pirmas vaikas yra pildomas elementais iš pirmojo tėvo. Jeigu elementas jau egzistuoja, jis yra pakeičiamas pagal gautus atitikimus pirmos dalies apkeitimo metu. Pavyzdžiui, primas vaiko 1-as elementas būtų 1, tačiau kadangi jis jau egzistuoja, reikia žiūrėti į atitikimus. Šiuo atveju užtenka vieno žingsnio, tai yra $4 \leftrightarrow 1$. Taigi, pirmojo vaiko 1-as elementas yra 4. Visus likusius, išskyrus paskutinį, galime pasiimti iš pirmojo tėvo. Paskutinio elemento atveju turime 8, kas jau egzistuoja pildomame vaike. Tokiu atveju atliekame 2 pakeitimus, $8 \leftrightarrow 6$ ir $6 \leftrightarrow 5$. Taigi, paskutinis elementas yra 5. Tada rezultatas atrodo šitaip:

Vaikas 1 (4 2 3 | 1 6 8| 7 5)

Analogiškai randame elementus ir antram vaikui, naudojant antro tėvo elementus.

Vaikas 2 (3 7 8 | 4 5 6 | 2 1)

7.3.2. Eilės numerio kryžminimas

Eilės numerio kryžminimo operacija [Dav85] (toliau OX) išnaudoją tai, kad svarbu yra viršūnių eiliškumas, o ne jų pozicija. Ši operacija sukonstruoja vaikus pasirinkdama vieno tėvo vidinę seką ir papildo kito tėvo viršūnėmis, kurios išlaiko reliatyvų eiliškumą. Pavyzdžiui, turi 2 tėvus, kurie buvo atsitiktinai atkirsti dviejose vietose (panašiai kaip PMX atveju).

Tėvas 1 (1 2 | 3 4 5 | 6 7 8)

Tėvas 2 (2 4 | 6 8 7 | 5 3 1)

Vaikai yra pradedami kurti šitaip:

Vaikas 1 (x x | 3 4 5 | x x x)

Vaikas 2 (x x | 6 8 7 | x x x)

Tada pradedant nuo antro atkirčio taško ir pereinant į pradžią, yra pildomi elementais iš kito tėvo, praleidžiant jau įterptus. Gauname tokį pirmąjį vaiką, praleidžiami elementus (3, 4 ir 5), visi kiti elementai išlaiko tvarką:

(8 7 | 3 4 5 | 1 2 6)

Analogiškai gauname ir antrąjį:

(4 5 | 6 8 7| 1 2 3)

7.3.3. Ciklinis kryžminimas

Ciklinio kryžminimo operacija [OSH87] (toliau CX) sukuria vaikus pagal duotus tėvus, kur kiekviena pozicija yra užimta pagal vieno iš tėvų atitinkamą elementą. Tarkime turime du tėvus:

Tėvas 1 (1 2 3 4 5 6 7 8)

Tėvas 2 (8 5 2 1 3 6 4 7)

Tada atsitiktinai pasirenkame pirmą vaiko elementą pagal vieną iš tėvų. Šiuo atveju jis gali būti 1 arba 8. Tegu būna 1.

(1 x x x x x x x)

Kiekvienas elementas turi būti pasirenkamas iš vieno iš tėvų. Dabar nėra kito pasirinkimo, nes 1 poziciją kitame tėve atitinka 8. 8 įrašome į atitinkamą poziciją pagal pirmąjį tėvą:

(1 x x x x x x 8)

Šio elemento pozicijoje yra 7, taigi analogiškai įrašome ir jį:

(1 x x x x x 7 8)

Toliau 7 poziciją atitinka 4:

(1 x x 4 x x 7 8)

Tada prieiname pirmojo ciklo galą, nes 4 pozicijoje yra 1, kas jau buvo įtrauktas. Analogiškai naudojame antrąjį tėvą elementams užpildyti:

(1 5 2 4 3 6 7 8)

Panašiai gauname ir antrąjį vaiką pradedami nuo kito tėvo (tai yra, jeigu pirmą vaiką pradėjome pildyti nuo pirmojo tėvo, antrą pradėti nuo atrojo):

(8 2 3 1 5 6 4 7)

Šios operacijos trūkumas yra tas, kad kartais ji sugeneruoja identiškus vaikus. Pavyzdžiui, jeigu turime tėvus:

Tėvas 1 (3 4 8 2 7 1 6 5)

Tėvas 2 (4 2 5 1 6 8 3 7)

Gauti vaikai bus identiški tėvams:

Vaikas 1 (3 4 8 2 7 1 6 5)

Vaikas 2 (4 2 5 1 6 8 3 7)

7.3.4. Implementacijos detalės

Šiame darbe eksperimentams yra naudojamas anksčiau apibendrintas GA. Pajėgumas yra tikslo funkcijos įvertis. Eksperimentams yra naudojamos visos operacijos (kryžminimo ir mutavimo) pagal toliau aprašytą parametrų lyginimo schemą, tokiu būdu randant kombinaciją kuri vidutiniškai atneša geriausią rezultatą.

7.4. Atkaitinimo modeliavimo pritaikymas

Naudojama ta pati sprendinio reprezentacija. Toliau aprašyta autoriaus siūloma atkaitinimo modeliavimo variacija. Šis algoritmas modeliuoja metalurgijos procesą, kada aukštos temperatūros metalą galima lengviau formuoti, nei žemos. Todėl temperatūros metafora savotiškai atitinka algoritmo iteracijas. Algoritmo metu yra naudojama artimiausio kaimyno godžioji euristika sudaryti pradinį sprendinį. Sprendinio kaimynystė yra gaunama naudojant vieną ar kelias minėtas mutacijos operacijas. Mutacijų intensyvumas priklauso nuo temperatūros, tokiu būdu padidinant paieškos erdvė algoritmo pradžioje ir visai sumažinant jo pabaigoje. Mutacijų intensyvumas yra įgyvendinamas tikimybę atlikti kelias mutacijas surišant su esama algoritmo temperatūros reikšme, tai yra, kuo didesnė temperatūra, tuo daugiau mutacijų bus atlikta. Eksperimentams yra naudojamos visos mutavimo operacijos pagal toliau aprašytą parametrų lyginimo schemą, tokiu būdu randant kombinaciją kuri vidutiniškai atneša geriausią rezultatą. Taip pat yra lyginamos temperatūros kitimo progresijos (nykstamoji geometrinė, tiesinė).

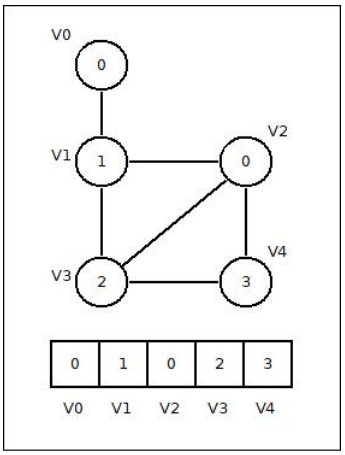
7.5. Skruzdžių kolonijos sistemos algoritmo pritaikymas

Naudojama ta pati sprendinio reprezentacija. Toliau aprašyta autoriaus siūloma skruzdžių kolonijos sistemos variacija. Euristiniam sprendiniui surasti yra naudojama ta pati artimiausio kaimyno godžioji euristika. Feromonai yra išsaugomi ant lankų kiekviename sprendinio komponente po lygiai. Feromonų kiekis vienai skruzdei priklauso nuo sprendinio gerumo. Tai yra, kuo trumpesnis kelias, tuo daugiau feromonų bus išsaugotą ant tą kelią sudarančių komponentų. Jeigu grafas yra simetriškas, tai feromonai yra tie patys, nepriklausomai iš kurios viršūnes lankas buvo aplankytas. Skruzdės ėjimo etape yra atsitiktinai pasirenkama strategija (artimiausio arba stipriausio feromono). Stipriausio feromono strategija yra atsitiktinai pasirenkamas lankas (jungus esamai viršūnei), o lanko tikimybė yra proporcingai nustatoma pagal feromonų kiekį. Yra lyginami globalaus (iki šiol geriausio) sprendinio feromonų įtakos bei lokalių sprendinių feromonų įtakos parametrų reikšmės. Taip pat lyginami strategijos pasirinkimo proporcijos bei feromonų garavimo greitis.

8. Grafo viršūnių spalvinimo uždavinio eksperimentų sąlygų konkretizavimas

Grafo spalvinimo uždavinys (angl. *Graph Coloring Problem*) (toliau GCP) yra rasti mažiausią viršūnių spalvų konfigūraciją jungiame grafe, kai nėra tokių lankų, kurių abi viršūnės būtų tos pačios spalvos. Yra sprendžiami uždaviniai iš COLOR [MS] duomenų bibliotekos: david, huck, queen8_8, myciel5, 1-FullIns_3 kurių yra žinomas optimalus sprendinys.

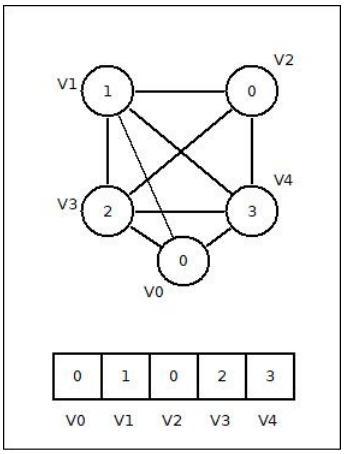
Grafo sprendinio reprezentacijos pavyzdys parodytas 8 pav.:



8 pav. GCP sprendinio reprezentacija. Viršūnės pozicijoje yra išsaugojama jai spalvinti naudojama spalva. Skirtingas skaičius atitinka skirtingą spalvą.

8.1. Grafo reprezentacijos papildas

Originalumo paieškoje (NS), yra reikalingas sprendinio reprezentacijos grafinis pavidalas, siekiant pritaikyti grafo originalumo funkcijas. Buvo sugalvotas ir taikomas tokiems atvejams grafo spalvų papildo grafas. Pagal sprendinio 8 pavyzdį, yra sukuriamas spalvų papildo grafas, apjungiant visas viršūnes, kurioms spalvinti yra naudojamos skirtingos spalvos. Tokiu būdu, yra gaunamas žymiai tankesnis grafas, tačiau jo tankumas priklauso nuo panaudojamų spalvų kiekio. Taip iš dalies yra išreiškiamas sprendinio gerumas spalvų papildo grafo lankų tankyje. Papildo grafas kinta nuo sprendinio, bet viršūnės išlieka tos pačios.



9 pav. GCP sprendinio spalvų papildo grafas. Apjungiamos viršūnės turinčios skirtingą spalvą.

8.2. Panašumo funkcijos

8.2.1. Tikslo funkcijos įverčio nuotolis

Kaip ir TSP atveju, primityviausia panašumo funkcija yra paremta tikslo funkcija. Jeigu sprendiniai turi tokius pačius (ar nedaug skiriančius) skirtingų spalvų kiekius, jie yra panašūs. Ši funkcija labai supaprastina ir todėl nėra tinkama originalumui įvertinti. Visgi, jeigu grafas yra pakankamai mažas, jos gali užtekti. Panašumo įvertis gaunamas iš spalvų kiekio skirtumo. Tai yra, kuo skirtumas mažesnis, tuo labiau sprendiniai panašūs.

8.2.2. Vienodų spalvų pozicijos

Ši panašumo funkcija vertina sutampančias spalvas atitinkamuose viršūnių pozicijose. Pavyzdžiui, jeigu turime du sprendinius, tai matome, kad paskutinės trys pozicijos sutampa.

(2 2 1 0 0 2 4 3)

(1 1 0 1 0 2 4 3)

Tada panašumo įvertį gauname taip:

$$f(C) = \frac{C}{S} \tag{7}$$

Čia C yra sutampančių spalvų pozicijų kiekis, o S yra sprendinio ilgis (statinis, nes visi sprendiniai vienodo ilgio).

8.2.3. Vienodas izomorfiškumas

Ši panašumo funkcija atrenka tos pačios spalvos rinkinius ir tikrina spalvinimo izomorfiškumą (kada sprendinys tas pats, tik skiriasi žymėjimas). Pilnai izomorfiškų sprendinių pavyzdys gali būti toks:

(2 2 1 0 0 2 4 3)

(3 3 2 1 1 3 0 4)

Tokiu atveju vienodų spalvų viršūnių rinkiniai sutaps. Kuo labiau izomorfiški grafai, tuo labiau jie yra panašūs. Tai nesunkiai galima padaryti taip:

1. Kiekvienam iš sprendinių sugrupuoti viršūnes pagal spalvą, taip gauname vienodų spalvų viršūnių rinkinius.

2. Turint šiuos rinkinius palyginti juos tarp kiekvieno iš sprendinių, išrenkant sutampančius rinkinius (visos viršūnės vieno sprendinio rinkinyje yra ir kito sprendinio rinkinyje.).

Tada panašumo įvertį gauname taip:

$$f(C,S) = \frac{C}{S} \tag{8}$$

Čia C yra sutampančių rinkinių kiekis, o S yra sprendinio didesnis spalvų kiekis.

8.3. Sprendinio lokalūs pokyčiai

GCP sprendinio reprezentacija sutampa visuose nagrinėjamuose algoritmuose. Tai yra viršūnių spalvų (skaičių) sąrašas, kurio pozicijos atitinka viršūnę. Analogiškai kaip ir TSP atveju, reikalingos mutacijos norint „vaikščioti“ po sprendinių erdvę, tačiau šiuo atveju nėra būtina išlaikyti legalų sprindinį, nes po kiekvienos mutacijos yra vykdomas konfliktų išsprendimas. Toks konfliktų išsprendimas yra visada įmanomas, nes naudojamų spalvų kiekis nėra ribojamas. Toliau išvardintos mutacijos yra naudojamos eksperimentiniuose tyrimuose.

8.3.1. Godžioji euristika DSatur

Grafo spalvinimo užduoties sprendimo metu yra plačiai naudojama DSatur godžioji euristika [Woo97]. Ji pasižymi tuo, kad rūšiuoja viršūnes pagal spalvų koncetraciją (angl. *Degree of saturation*)

1. Pasirinkti nenuspalvintą viršūnę V su didžiausiu unikaliai (jeigu yra dvi gretimos viršūnės tos pačios spalvos, skaičiuojama kaip viena) nuspalvintų viršunių kiekiu

2. Jeigu yra kelios tokius viršūnės, pasirinkti pirmąją su didžiausiu laipsniu

3. Jeigu nebėra nenuspalvintų viršūnių, baigti algoritmą

4. Priskirti V viršūnei pirmąją tinkamą (nesutampančią su kaimyninėmis) spalvą

8.3.2. Nelegalios mutacijos ištaisymas

Taip pat nesunku pastebėti, kad galima naudoti ir TSP atvejui aprašytas mutacijas, tačiau jos dažnai gauna nelegalų spalvų rinkinį (kada gretimos viršūnės turi tą pačią spalvą). Tokiu atveju, reikia pataisyti sprendinį. Visos konfliktuojančios viršūnės yra sugrupuojamos į skaidinį ir perdažamos pagal euristinį algoritmą DSatur.

8.4. Genetinio algoritmo pritaikymas

GA metu yra svarbi sprendinio reprezentacija, kurią galima būtų skaldyti. Esanti reprezentacija, kai viršūnių pozicijose yra išsidėsčiusios spalvos, yra tinkama. Svarbesnė GA dalis yra kryžminimas, tačiau tai įgyvendinti išlaikant legalų sprendinį yra žymiai sunkiau ne TSP atveju. Žinoma, galima ignoruoti sprendinio netaisyklingumą ir jį prireikus jį ištaisyti.

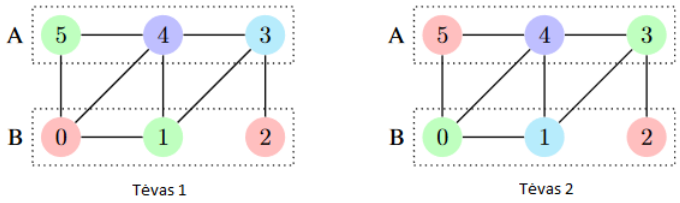
Visgi, yra keletas kryžminimo operacijų, kurios yra naudojamos genetinio algoritmo metu:

- Pografių kryžminimas (angl. *Partition crossover*) (PartX)
- Pografius atskiriantis kryžminimas (angl. *Separator crossover*) (SepX)
- Vieno atkirtos taško kryžminimas (angl. *Single point crossover*) (SPX)
- Dviejų atkirtos taškų kryžminimas (angl. *Two point crossover*) (TPX)

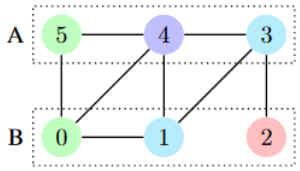
Visi konfliktai kryžminimo metu yra sprendžiami naudojant DSatur algoritmą, prieš tai pažymint abi viršūnes dalyvaujančias konflikte kaip nenuspalvintas.

8.4.1. Skaidinių kryžminimas

Šis kryžminimo būdas [Mum06] suskaido grafo viršūnes į 2 skaidinius A, B, kaip parodyta 10. Šitie skaidiniai yra tokio pat dydžio (nelyginiu viršūnių atveju A yra viena didesnis).



10 pav. Skaidiniai A, B yra įrėminti, kurie naudojami kryžminimo operacijoje

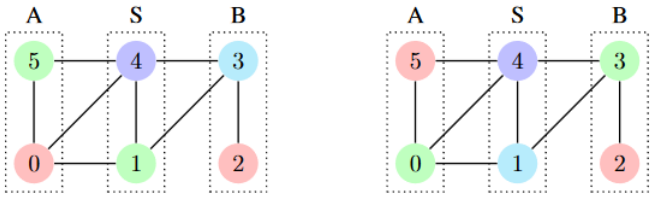


11 pav. Vienas iš gaunamų vaikų, prieš ištaisant spalvų konfliktą

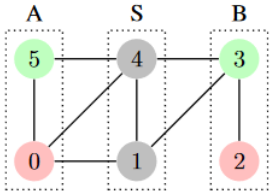
Kadangi tėviniai sprendiniai yra taisyklingi, jeigu viršūnė ir visi jos kaimynai yra tame pačiame pografyje, tada ši viršūnė negali sudaryti konflikto. Tačiau, bet koks lankas jungiantis šiuos skaidinius gali sudaryti konfliktą, kaip ir gaunasi 11.

8.4.2. Pografius atskiriantis kryžminimas

Šis kryžminimo būdas [Mum06] yra panašus į pografių kryžminimą, tačiau naudoja pografių atkyrimą per susijusias viršūnes. Kryžminimo metu viršūnės yra suskaidomos į 3 pografius: A,B,S. Pografiai A, B yra parenkami panašaus dydžio, kurių viršūnės neturi bendrų lankų, o S pografis yra atskiriamasis. Tai yra, jame yra visos viršūnės, kurios negali būti nei A, nei B grafe. Kryžminimas prasideda visą grafą priskiriant pografiui S, ir po vieną viršūnę perkeliant į likusius A, B pografius, išlaikant jungumą. Pradinės viršūnės, kurios bus įtrauktos į A ir B pografius, yra pasirenkamos grafo taip, kad tarp jų butų didžiausias atstumas. Toks procesas daromas vienam iš tėvų ir gauti pografiai pritaikomi abiems tėvams, kaip parodyta pavyzdyje 12



12 pav. Pografiai A, B yra įrėminti, kurie naudojami kryžminimo operacijoje, o S pografis lieka nenuspalvintas



13 pav. Vienas iš gaunamų vaikų, prieš S pografio nuspalvinimą

Po tokio kryžminimo reikalinga euristika, kuri nuspalvoja likusį grafą. Yra naudojama jau minėta euristika DSatur.

8.4.3. Vieno atkirtos taško kryžminimas

Tai yra paprastas vieno atkirtos taško kryžminimas [Lid91]. Pavyzdžiui turime tokius 2 sprendinius, kuriems yra atsitiktinai parinktas atkirtos taškas.

(2 2 1 0 | 0 2 4 3)

(1 1 0 1 | 3 2 1 0)

Tada atsitiktinai po kryžminimo operacijos gaunami vaikai yra tokie:

(2 2 1 0 | 3 2 1 0)

(1 1 0 1 | 0 2 4 3)

Jeigu gautas veikas nėra teisingas, konfliktuojančios viršūnės yra ištaisomos godžios euristikos DSatur algoritmu.

8.4.4. Dviejų atkirtos taškų kryžminimas

Tai yra panašus į vieno atkirtos taško kryžminimą [Lid91], tik jis skiriasi tuo, kad naudoja 2 atkirtos taškus. Pavyzdžiui turime tokius 2 sprendinius, kuriems yra atsitiktinai parinktas atkirtos taškas.

(2 2 1 | 0 0 2 | 4 3)

(1 1 0 | 1 3 2 | 1 0)

Tada atsitiktinai po kryžminimo operacijos gaunami vaikai yra tokie:

(2 2 1 |0 0 2 |1 0)

(2 2 1 |1 3 2 |4 3)

(1 1 0 |0 0 2 |4 3)

(1 1 0 |1 3 2 | 4 3)

Šis kryžminimo būdas leidžia sukonstruoti daugiau vaikų, tačiau taisyklingų sprendinių kiekis nebūtinai yra didesnis. Konfliktai sprendžiami analogiškai, kaip ir vieno atkirtos taško kryžminimo metu.

8.4.5. Implementacijos detalės

Genetinio algoritmo implementacija mažai skiriasi nuo TSP atveju naudojamos. Yra naudojamos visos išvardintos GCP tinkamos mutacijos ir kryžminimo operacijos, tokiu būdu randant kombinaciją ar operacijų santyki kuris vidutiniškai atneša geriausią rezultatą. Visa kita algoritmo implementacija sutampa su TSP atveju aprašyta.

8.5. Atkaitinimo modeliavimo pritaikymas

Algoritmo metu yra naudojama artimiausio kaimyno godžioji euristika DSatur sudaryti pradinį sprendinį. Kadangi GCP atveju dauguma mutacijų nėra iš karto pelningos (dažniausiai padidėja spalvų kiekis), mutacijų kiekis ir įvairovė turi būti didesnis nei TSP atveju. Sprendinio kaimynystė yra gaunama atliekant kelias minėtas mutacijos operacijas, kelis kartus, priklausomai nuo temperatūros. Visi kiti parametrų lyginimai yra analogiškai kaip ir TSP atveju.

8.6. Skruzdžių kolonijos sistemos algoritmo pritaikymas

Naudojama ta pati sprendinio reprezentacija. Pradiniam euristiniam sprendiniui surasti yra naudojama ta pati godžioji euristika DSatur. Yra formuojama viršūnių seka, kurią sekant turi būti

nuspalvinamas grafas. Feromonai yra išsaugojami ant viršūnių. Jie daro įtaką kitos viršūnės pasirinkimui. Pasirinkimas yra daromas pasirenkant iš nenuspalvintų viršūnių atsitinktinai naudojant arba DSatur arba feromonų strategiją. Feromonų strategija yra tokia:

- Kuo anksčiau viršūnių priskirimo sekoje yra viršūnė, tuo daugiau feromonų joje bus išsaugota.
- Kuo daugiau sprendinys turi spalvų, tuo mažiau feromonų bus išsaugota.

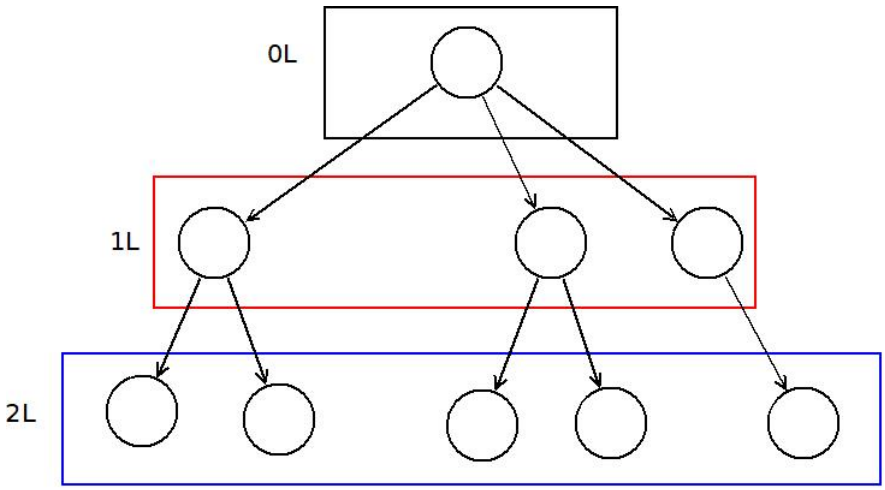
Likusi implementacija ir palyginimai yra vykdoma analogiškai kaip ir TSP atveju.

9. Originalumo paieškos pritaikymas

Originalumo paieška (NS) yra įgyvendinama pakeičiant genetinio algoritmo tikslo funkciją originalumo įverčiu. Originalumo įverčiai gaunami naudojant panašumo funkcijas aprašytas anksčiau atitinkamam uždaviniui. Nustatant tam tikrą ribą, yra kaupiami sprendiniai, kurie yra sutapatinami su elgesiu. Eksperimentams yra naudojamos visos išvardintos kryžminimo ir mutavimo operacijos. Taip pat yra lyginama skirtingų archyvo plėtimo strategijų įtaka algoritmo efektyvumui. Po kiekvienos generacijos, visi individai populiacijoje yra patikrinami su išorine tikslo funkcija (TSP atveju tai kelionės kaina, o GCP atveju tai spalvų kiekis ir sprendinio teisingumas). Šis išorinis pajėgumo patikrinimas surenka gautus sprendinius, kurie pateikiami algoritmo pabaigoje. Tokiu būdu turėtų būti gaunami keli, pajėgūs sprendiniai, kurie žymiai skiriasi vienas nuo kito, jeigu algoritmas suveikė teisingai.

9.1. Hierarchinis sprendinių panašumo palyginimas

Vienas iš didžiausių algoritmo stabdžių yra archyvo kaupimas. Pradinės implementacijos matu sprendinių archyvas yra paprastas sąrašas, todėl naujo sprendinio originalumo patikrinimas turi būti padarytas su kiekvienu sprendinių esančiu archyve. Algoritmo eigoje natūralus archyvo augimas, todėl kas kartą šis patikrinimas ilgės. Kadangi algoritmas ir taip yra euristinis, rezultatams netūrėtų pakenkti archyvo euristinės optimizacijos.

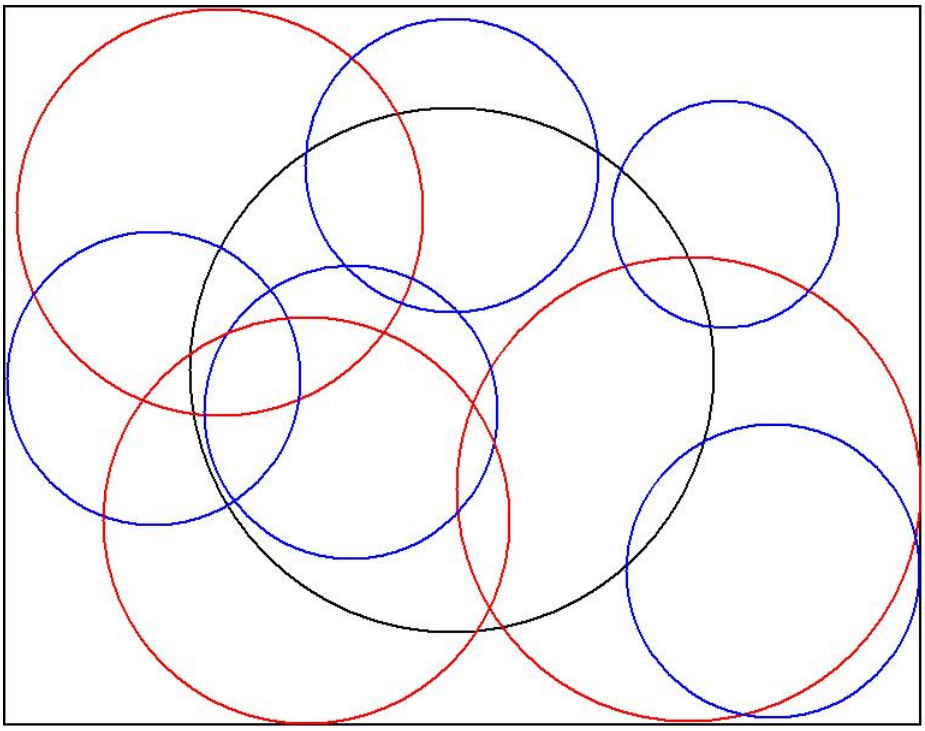


14 pav. Archyvo hierarchijos medis.

Archyvas yra suskirstomas yra lygius 14. Pradinis lygis (0L) yra sudarytas iš vieno sprendinio, kuris yra laikomas medžio šaknimi. Kiekvienas lygis turi savo nepanašumo ribą θ_L, kuri nulemia ar sprendinys pakankamai skiriasi nuo visų šiame lygyje esančių sprendinių, kad butų sukurta nauja šaka. Ši riba priklauso nuo gylio ir eksperimentiniu būdu parinkto parametrų β ir α. β nulemia kiek įtakos lygio gilumas daro nepanašumo ribai ir netiesiogiai nustato kiek lygių gali būti (kiek kartų galima atlikti operaciją $\alpha - \beta$), o α yra pradinė nepanašumo riba, kuri ir yra keičiama pagal lygį. Galutinė lygio nepanašumo riba yra tokia $\theta_L = \alpha - L \times \beta$. Tada archyvo algoritmas yra toks gavus naują sprendinį s (daroma prielaida, kad aibėje jau yra šaknis, todėl praleidžiame 0 lygį):

1. Pereiti į 1 lygį L, tėvinis sprendinys yra p
2. Palyginti s su visais C aibės sprendiniais, kurie yra p vaikai
3. Jeigu C yra tuščia arba C aibės mažiausiai panašus sprendinys m peržengia θ_L ribą, įtraukti s į C aibę ir baigti
4. Priešingu atveju, C aibės mažiausiai panašus sprendinys m tampa nauju tėviniu sprendiniu p, peržengiama į sekantį lygį $L + 1$ ir tęsiama nuo 2-o žingsnio.

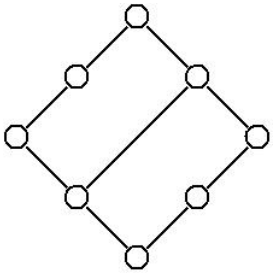
Toks algoritmas euristiškai grupuoja sprendinius pagal panašumą ir apriboja panašumų palyginimą bei archyvo plėtimą. Gaunamas savotiškas sprendinių hierarchinis negriežtas klasterizavimas (angl. *Hierarchical fuzzy clustering*) (žr. 15 pav.). Klasterio centrą atitinka tam tikras sprendinys, kuris atstovauja savo lygį atitinkamoje hierarchijos atšakoje. Kuo aukštesnis lygis, tuo labiau panašūs tos atšakos sprendiniai, tačiau tuo labiau nepanašūs skirtingų atšakų sprendiniai. Nors to paties lygio sprendiniai skirtingose atšakose gali būti panašūs tarpusavyje, jie nebus panašūs į visus prieš tai ėjusių lygių šakos atstovus. Kitaip tariant, neįveiks panašumo testo leidžiantis kitoje hierarchijos atšakoje.



15 pav. Archyvo hierarchijos medžio klasterizavimo sprendinių pasiskirstymo teorinis eskizas. Jeigu stačiakampio plotas yra visi įmanomi sprendiniai, tai klasteriai žymi kiek tas klasteris aprėpia sprendinių pagal panašumą. Taip pat atitinkama spalva žymi kurio lygio yra klasteris (spalvos sutampa su 14 pav.). Taip pat nesunku pastebėti klasterių ribų persikirtimą, tai simbolizuoja, kad tas pats sprendinys panašus į kelis klasterių lyderius, nors yra priskirtas tik vienam.

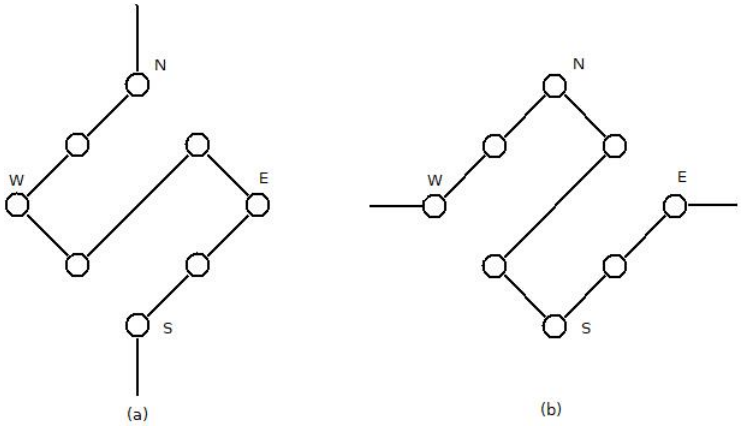
10. Sunkiai sprendžiamų keliaujančio uždavinių konstravimas

Siekiant išsamiai ištirti NS algoritmo pajėgumą ir pagrindinę hipotezę, kuri išvengia užstrigimo lokaliame ekstremume, tenka ieškoti specifinių grafų, kurie būtų ypač sudėtingi standartiniams (iteratyvaus gerinimo) euristiniems algoritmams. Tokia sukonstruotų grafų klasė yra aprašoma [PS78] darbe. Yra sukuriama deimanto klasė žr. 16 pav.



16 pav. Deimanto klasės grafas

Yra du būdai optimaliai (mažiausiai kainuojantis kelias) pereiti per deimanto klasės grafo viršūnes: Šiaurės-pietų (a) ir vakarų-rytų (b), kaip parodyta žemiau 17 pav. Tada galime sukurti



17 pav. Deimanto klasės grafo optimalus apėjimas

visą deimanto grafų klasę $G(k)$, kur k yra deimanto formos apjungtų pografių kiekis:

1. Sukurti k deimanto pografio kopijų $D_i, i = 1, \ldots, k$. Šiaurinės viršūnės atitinkamuose deimantuose pavadinti $N1$, vakarines $W1$, pietines $S1$ ir rytines $E1$. Sujungti rytines viršūnes su joms gretimo deimanto pografio vakarinėmis. Paskutinio pografio rytinę viršūnę su pirmo pografio vakarine. Tokio būdu gaunamas Hamiltono ciklas (kada apeinama per visas grafo viršūnes vieną kartą ir grįžtama į pradinę), kuris šiuo atveju yra vakarinis-rytinis ciklas. Kiekvieno ciklo lankui priskirkime kainą 1. Tada lieka kiekviename deimantiniame pografyje po du lankus be kainos. Jiems duokime kainą 0.

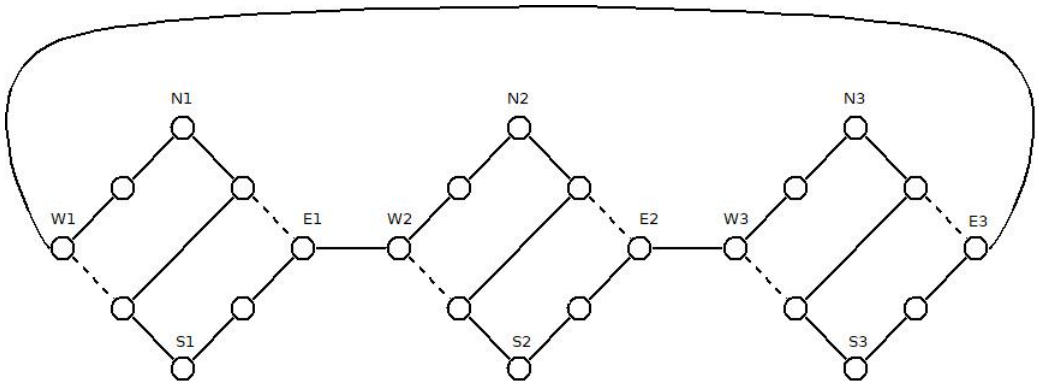
   Toliau idėja yra pridėti daug 0 kainos lankų, kurie apjungia šiaurines su pietinėmis viršūnėmis, bet ir tuo pačiu trukdo optimaliai pereiti šiaurės-pietų rėžimu. Tai yra įgyvendinama

izoliuojant viršūnes, pavyzdžiui $N1$, apjungiant ją su kitomis šiaurinėmis ir pietinėmis viršūnėmis didžiulę kainą turinčiais lankais.

2. Apjungti visą 2k-1 aibę viršūnių $A = \{N2,N3,\ldots,Nk,S1,\ldots,Sk\}$ su 0 kainos lankais. Tada sujungti likusią viršūnę $N1$ su visomis kitomis viršūnėmis aibėje A su M kainos lankais. M yra vartotojo pasirinkta aukšta kaina (pvz. 1000).

3. Kiekvieną viršūnių porą, kuri dar nėra apjungta, apjungti su lanku, kurio kaina yra $2M$.

Pagrindinė savybė tokio $G(k)$ TSP atveju yra tai, kad egzistuoja viena optimali kelionę, kurios kaina yra $8k$, gaunama pereinant $G(k)$ grafą vakarų-rytų Hamiltono ciklu, kaip parodyta 18 pav. Kiti geriausi ciklai, kurių yra $2^{k-1}(k-1)!$ kainuoja $M+5k$ ir skiriasi nuo optimalaus ciklo $3k$ lankais. Įrodymas yra aprašytas [PS78].



18 pav. Deimanto klasės $G(3)$ grafo optimalus vakarų-rytų tipo apėjimas

Deja, sudėtingo grafo konstravimo, kuris veiktu bendru atveju nėra. Yra sunkiai spalvinami grafai konkretaus algoritmo atveju (pvz. DSatur), bet bendru inkrementinio gerinimo atveju tokio grafp konstravimo algoritmo nėra.

11. Parametrų lyginimo schema ir rezultatų formavimas

Euristiniai algoritmai pasižymi rezultatų kaita dėl determinizmo trūkumo. Dėl šios priežasties kiekviena parametrų konfigūracija bus vykdoma 5 kartus.

Parametro lyginimo detalumas yra 20%. Pavyzdžiui jeigu parametras gali įgyti reikšmes nuo 0 iki 1, jis įgis šias reikšmes (0; 0,20; 0,40; 0.60; 0,80; 1).

Parametrų lyginimo tikslas yra nustatyti kaip koks parametras veikia rezultatą bei rasti optimalią (ar bent jau artimą optimaliai) parametrų konfigūraciją konkrečios užduoties aplinkos kontekste. Parametrų lyginimas prasidės nuo pagal nutylėjimą (angl. *default*) gautos kombinacijos, kas šiuo atveju yra 50% kiekviename parametre. Tada iš eilės pagal detalumo lygį bus keičiamas pirmas parametras, po to antras ir taip toliau. Po pirmos parametrų lyginimo epochos, pradinė konfigūracija yra keičiama pagal tas reikšmes, kurių atveju rezultatas buvo geriausias. Tada analogiškai pagal minėtą detalumo lygį yra tokia pat tvarka keičiami parametrai. Jeigu epochos geriausia parametrų kombinacija yra ne vienintelė, geriausia yra laikoma anksčiau aptikta. Pavyzdžiui, jeigu turime 2 keičiamus parametrus a, b, kurie įgyja reikšmes nuo 0 iki 1, tai tada parametrų kaita bus tokia (žr. 1):

Epoch.-Iter.	a	b	Rezultatas
1-1	0	0.5	1
1-2	0.2	0.5	2
1-3	0.4	0.5	3
1-4	0.6	0.5	2
1-5	0.8	0.5	3
1-6	1	0.5	2
1-7	0.5	0	2
1-8	0.5	0.2	1
1-9	0.5	0.4	2
1-10	0.5	0.6	3
1-11	0.5	0.8	4
1-12	0.5	1	1
2-1	0	0.8	2
2-2	0.2	0.8	3
2-3	0.4	0.8	4
2-4	0.6	0.8	3
2-5	0.8	0.8	4
2-6	1	0.8	3
2-7	0.4	0	3
2-8	0.4	0.2	2
2-9	0.4	0.4	3
2-10	0.4	0.6	6
2-11	0.4	0.8	5
2-12	0.4	1	2

1 lentelė. Parametrų kaitos pavyzdys. Iš pirmos epochos geriausias rezultatas 1-3 ir 1-11 iteracijose, tada naudojant tuos parametrus $a = 0.4, b = 0.8$ yra gaunama nauja rezultatų epocha. Po antros epochos yra randama geriausia parametrų kombinacija 2-10 atveju, kai $a = 0.4$ ir $b = 0.6$.

Tokiu būdu yra ženkliai sumažinamas lyginamų parametrų kombinacijų kiekis ir yra tikslingai gaunamos įžvalgos į konkretaus parametro įtaką. Taip pat kaip ta įtaka kinta, kai yra naudojamos kitos mutacijos (ar kiti algoritmai GA atveju).

Radus geriausią parametrų kombinaciją, eksperimentas su ja yra kartojamas dar 5 kartus (iš viso 10 kartų).

Mutacijos yra kombinuojamos taip:

- Kiekviena iš mutacijų atskirai
- Visos mutacijos, kai kiekvieną pasirinkti yra vienoda tikimybė

Kryžminimai nėra kombinuojami, tačiau su kiekvienu iš jų reikia atlikti anksčiau išvardintus veiksmus.

11.1. Rezultatų formavimas

Euristiniai algoritmai iš prigimties yra atsitiktiniai, todėl norint tikslingai įvertinti algoritmo efektyvumą, reikia tą pačią užduotį spręsti daug kartų ir matuoti vidutinį sprendinį. Abiejų uždavinių aplinkos yra kiek įmanoma labiau suvienodintos kiekvienam algoritmui. Taip pat lyginamos to paties algoritmo skirtingų parametrų konfigūracijos pagal minėtą parametrų lyginimo schemą. Tada yra išrenkama palankiausia algoritmo parametrų konfigūracija, todėl yra užtikrinama sąžiningiausios sąlygos, kiekvieno algoritmo atžvilgiu. Algoritmai yra lyginami su kitų algoritmų palankiausiomis konfigūracijomis. Tikslo funkcijų įverčių kiekis yra suvienodinamas kiek įmanoma daugiau, bei yra fiksuojamas užduoties išsprendimų kiekis (ar algoritmas užduotį išsprendžia pastoviai).

12. Rezultatai

12.1. Palankiausios algoritmų aplinkos

12.1.1. Genetinis algoritmas (GA)

Genetinio algoritmo TSP tyrimo metu buvo naudojamos 3 skirtingos TSP tipo kryžminimo operacijos ir 5 skirtingos mutacijos operacijos (4 atskiros ir 1 kada atsitiktinai pasirenkama viena iš visų) Geriausi sprendiniai buvo gauti naudojant kelio dalies nukirtimo ir regeneravimo mutaciją bei dalinai perduodamą kryžminimo mutaciją. Mutacijos bei kryžminimo santykis neturėjo įtakos. Populiacijos kiekis turėjo įtakos, nes kuo daugiau, tuo „plačiau“ vyksta paieška. Par mažos populiacijos dydis (20 ir mažiau) stabdė procesą (buvo rastas prastesnis geriausias sprendinys), nes neužteko sprendinių įvairovės, o per didelis (virš 100) jo negreitino, nes tai tiesiog padidino funkcijos įverčių kiekį, bet nepagerina galutinio sprendinio. Iteracijų kiekis neturėjo įtakos kai viršija 200, nes dažniausiai jau būna pasiektas nekintantis sprendinys.

GC tyrimo metu, buvo naudotos 4 skirtingos GC tipo kryžminimo operacijos ir tos pačios kaip ir TSP atveju mutacijos su konfliktų ištaisymu DSatur algoritmo pagalba. Geriausi sprendiniai buvo gauti naudojant Maksimali stagnacija dažnai viršydavo 500, kas reiškia, kad per 500 generacijų nebuvo rastas geresnis maršrutas (TSP atveju) ar spalvinimas (GC atveju).

12.1.2. Skruzdžių kolonijos sistema (ACO)

Skruzdžių kolonijos tyrimo metu nebuvo naudojamos mutacijos ar kryžminimai, bet tiriami parametrai. TSP atveju buvo atrasta, kad palankiau yra naudoti feromonais pagrįstą strategiją, nei godžią, tačiau GC atveju DSatur algoritmas pasirodė esąs pranašesnis. Mažas feromonų nykimas lėmė greitą sprendinių konvergaciją (suvienodėjimą) ir mažą įvairovę, nes pradinis geriausias sprendinys dominavo visos algoritmo sekos metu. Didesnis skruzdžių kiekis lėmė didesnę sprendinių įvairovę algoritmo pradžioje (skruzdė kelią ar seką pradeda konstruoti nuo atsitiktinos viršūnės), tačiau ir didesnį tikslo funkcijos įverčių kiekį. Lokalių feromonų įtakos kiekis lėmė lėtesnę sprendinių konvergaciją.

12.1.3. Atkaitinimo modeliavimas (SA)

Atkaitinimo modeliavimo tyrimo metu buvo naudojamos visos 5 mutacijos. Mutacijos intensyvumas (kartai) neturėjo įtakos, nes sprendinių įvairovė algoritmo pradžioje nepakito. Temperatūros progresijos nulėmė iteracijų kiekį be tikimybę priskirti blogesnį sprendinį vietoj esančio. Tiesinė progresija buvo palankesnė, nes algoritmo pabaigoje dažniau priskirdavo blogesnį sprendinį tokiu būdu išvengdama lokalaus ekstremumo. Žinoma, kuo mažiau temperatūra mažėja, tuo daugiau iteracijų, todėl ir geresnis sprendinys buvo rastas, bet tai dažniausiai nebesikeitė iteracijų kiekiui peršokus apie 200.

12.1.4. Originalumo paieška (NS)

Originalumo paieškos metu buvo naudojamos visi kryžminimai bei mutacijos kaip ir tos pačios užduoties genetinio algoritmu metu. TSP ir GC atveju geriausi sprendiniai buvo gaunami naudojant visų mutacijų kombinaciją. Tai nutiko dėl to, kad originalumo paieškoje svarbiausia yra sprendinių įvairovė, todėl keičiant mutacijas ir yra užtikrinama didesnė sprendinių įvairovė. Kryžminimai neturėjo didelės įtakos, nes mutacijos buvo pagrindinis įvairovės šaltinis. TSP atveju geriausia panašumo funkcija buvo sutampančių lankų kiekio, GC atveju - vienodos spalvų pozicijos.

Hierarchinį ir sąrašo tipo archyvai neturėjo įtakos sprendinių kokybei, tačiau hierarchinis sumažino panašumo funkcijų įverčių kiekį. Priešingai nei GA atveju, populiacijos dydis neturėjo įtakos, nes sprendinių

12.2. Konkrečių uždavinių ir algoritmų rezultatai

Apačioje pateikta TSP ir GC uždavinio algoritmų geriausių parametrų rezultatai. Čia |V| yra viršūnių kiekis, |E| yra lankų kiekis, Rbest yra geriausias algoritmo rastas rezultatas, Ropt yra žinomas optimalus rezultatas, Fopt yra kiek kartų iš 10 buvo rastas optimalus rezultatas.
TSP grafai:att48, berlin52, bays29, G(4), G(8). Čia G(4) ir G(8) yra deimanto klasės generuojami ir sunkiai sprendžiami grafai.
GC grafai:david, huck, queen8_8, myciel5, 1-FullIns_3. GA, NS, ACS atveju iteracijų kiekis yra 1000, o SA atveju jis apytiksliai siekia 1000 (priklauso nuo temperatūros kitimo strategijos).

Alg.	Grafas	|V|	|E|	Rbest	Ropt	Fopt
GA	G(4)	24	552	1020	24	0
	G(8)	64	4032	1032	64	0
	berlin52	52	2652	7542	7542	8
	att48	48	2256	33522	33522	9
	bays29	29	812	2020	2020	10
ACO	G(4)	24	552	1020	24	0
	G(8)	64	4032	1032	64	0
	berlin52	52	2652	7542	7542	9
	att48	48	2256	33522	33522	10
	bays29	29	812	2020	2020	10
SA	G(4)	24	552	1020	24	0
	G(8)	64	4032	1032	64	0
	berlin52	52	2652	7542	7542	5
	att48	48	2256	33522	33522	5
	bays29	29	812	2020	2020	9
NS	G(4)	24	552	24	24	2
	G(8)	64	4032	1032	64	0
	berlin52	52	2652	8995	7542	0
	att48	48	2256	42753	33522	0
	bays29	29	812	2020	2020	2

2 lentelė. Keliaujančio pirklio uždavinio tyrimo geriausių parametrų algoritmų rezultatų palyginimas

Alg.	Grafas	|V|	|E|	Rbest	Ropt	Fopt
GA	david	87	406	11	11	6
	huck	74	301	13	13	8
	queen8_8	64	728	9	9	7
	myciel5	47	236	6	6	8
	1-FullIns_3	30	100	4	4	10
ACO	david	87	406	12	11	4
	huck	74	301	14	13	4
	queen8_8	64	728	9	9	5
	myciel5	47	236	6	6	7
	1-FullIns_3	30	100	4	4	10
SA	david	87	406	12	11	5
	huck	74	301	13	13	7
	queen8_8	64	728	9	9	5
	myciel5	47	236	6	6	5
	1-FullIns_3	30	100	4	4	10
NS	david	87	406	14	11	0
	huck	74	301	15	13	0
	queen8_8	64	728	12	9	0
	myciel5	47	236	6	6	2
	1-FullIns_3	30	100	4	4	4

3 lentelė. Grafo spalvinimo uždavinio tyrimo geriausių parametrų algoritmų rezultatų palyginimas

13. Išvados

Šiame darbe buvo nagrinėjami euristiniai algoritmai, skirti spręsti dviejų pobūdžių grafų optimizavimo uždavinius: Keliaujančio pirklio ir grafo spalvinimo. Formalizuota nauja euristika - originalumo paieška, kurios idėja atėjo iš visai kitos kompiuterių mokslo srities. Deja, ši euristika nėra efektyvi bendro grafo pavidalo atveju:

- Įmanomų sprendinių aibė yra per didelė, todėl yra per daug laiko skiriama nevaisingų sprendinių kūrimui.
- Kartais yra priartėjama prie optimalaus sprendinio, tačiau tai atsitinka retai ir nepastoviai.
- Reikalingas didžiulis iteracijų kiekis, siekiant pakankamai ištirti sprendinių aibę.
- Ištirti iteratyvaus gerinimo euristiniai algoritmai turi pakankamai įvestos įvairovės, kad gana gerai (pastoviai ir arti optimaliai) išspręstu uždavinius nedideliuose grafuose.
- Vieninteliu atveju, kada yra specialiai generuojamas deimanto (žr. 16 pav.) klasės grafas, NS euristika pralenkia tradicinius algoritmus.
- Panašumo funkcija yra naudojama labai dažnai (daugiau nei 10 kartų daugiau ne tikslo funkcija), todėl jos greitis nulemia viso algoritmo greitį.

NS euristikai yra pasiūlytas ir patikrintas naujas sprendinių archyvavimo metodas (archyvo hierarchijos medis), kuris sumažina panašumo funkcijos įverčių kiekį ir suteikia archyvui struktūros. Taip pat aptiktas vienas iš NS euristikos privalumų - jai nebūtinas didelis populiacijos kiekis. Grafo optimizavimo uždaviniams ši euristika nėra rekomenduojama. Buvo išsiaiškinta, kad neributo

Priedai

Visa implementacija gali būti pasiekiama GitHub adresu `https://github.com/laim0nas100/NoveltySearchGraph`

Literatūra

[AY05] Charu C Aggarwal ir Philip S Yu. Online analysis of community evolution in data streams. *Proceedings of the 2005 SIAM International Conference on Data Mining*, p.p. 56–67. SIAM, 2005.

[AMF10] L. Akoglu, M McGlohon ir C. Faloutsos. Oddball: spotting anomalies in weighted graphs. PAKDD Conference, 2010.

[Ban90] Wolfgang Banzhaf. The "molecular" traveling salesman. *Biological Cybernetics*, 64(1):7–14, 1990.

[BHK+06] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg ir Xiangyang Lan. Group formation in large social networks: membership, growth, and evolution. *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, p.p. 44–54. ACM, 2006.

[Bro11] Jason Brownlee. *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee, 2011.

[CAg16] Charu C.Aggarwal. *Outlier Analysis, Second Edition*. Springer publisher company, 2016.

[CH00] Diane J Cook ir Lawrence B Holder. Graph-based data mining. *IEEE Intelligent Systems and Their Applications*, 15(2):32–41, 2000.

[DAF+10] Pedro OS Vaz De Melo, Leman Akoglu, Christos Faloutsos ir Antonio AF Loureiro. Surprising patterns for the call duration distribution of mobile phone users. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, p.p. 354–369. Springer, 2010.

[Dav85] Lawrence Davis. Job shop scheduling with genetic algorithms. *Proceedings of an international conference on genetic algorithms and their applications*, tom. 140, 1985.

[Deb99] Kalyanmoy Deb. Multi-objective genetic algorithms: problem difficulties and construction of test problems. *Evolutionary computation*, 7(3):205–230, 1999.

[FP+99] Tom Fawcett, Foster J Provost ir k.t. Activity monitoring: noticing interesting changes in behavior. *KDD*, tom. 99, p.p. 53–62. Citeseer, 1999.

[GAH+11] Manish Gupta, Charu C Aggarwal, Jiawei Han ir Yizhou Sun. Evolutionary clustering and analysis of bibliographic networks. *2011 International Conference on Advances in Social Networks Analysis and Mining*, p.p. 63–70. IEEE, 2011.

[GAH11] Manish Gupta, Charu C Aggarwal ir Jiawei Han. Finding top-k shortest path distance changes in an evolutionary network. *International Symposium on Spatial and Temporal Databases*, p.p. 130–148. Springer, 2011.

[GG09] Atul P Godse ir Deepali A Godse. *Computer Organization*. Technical Publications, 2009.

[GH88] David E Goldberg ir John Henry Holland. Genetic algorithms and machine learning, 1988.

[GL+85] David E Goldberg, Robert Lingle ir k.t. Alleles, loci, and the traveling salesman problem. *Proceedings of an international conference on genetic algorithms and their applications*, tom. 154, p.p. 154–159. Lawrence Erlbaum, Hillsdale, NJ, 1985.

[GR+87] David E Goldberg, Jon Richardson ir k.t. Genetic algorithms with sharing for multimodal function optimization. *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, p.p. 41–49. Hillsdale, NJ: Lawrence Erlbaum, 1987.

[HL06] Marcus Hutter ir Shane Legg. Fitness uniform optimization. *IEEE Transactions on Evolutionary Computation*, 10(5):568–589, 2006.

[LEH12] JOEL LEHMAN. *EVOLUTION THROUGH THE SEARCH FOR NOVELTY*. Disertacija, University of Central Florida Orlando, Florida, 2012.

[Lid91] ML Lidd. Traveling salesman problem domain application of a fundamentally new approach to utilizing genetic algorithms. *Research sponsored in part by Air Force Office of Scientific Research and Office of Naval Research, Contract F4920-90-G-0033*, 1991.

[Lyn07] Michael Lynch. The frailty of adaptive hypotheses for the origins of organismal complexity. *Proceedings of the National Academy of Sciences*, 104(suppl 1):8597–8604, 2007.

[LS08a] Joel Lehman ir Kenneth O Stanley. Exploiting open-endedness to solve problems through the search for novelty. *ALIFE*, p.p. 329–336, 2008.

[LS08b] Joel Lehman ir Kenneth O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. *Proceedings of the Eleventh International Conference on Artificial Life (ALIFE XI)*, Cambridge. MIT Press, 2008.

[Mah95] Samir W Mahfoud. *Niching methods for genetic algorithms*. Disertacija, Citeseer, 1995.

[MJK92] Zbigniew Michalewicz, Cezary Z Janikow ir Jacek B Krawczyk. A modified genetic algorithm for optimal control problems. *Computers & Mathematics with Applications*, 23(12):83–94, 1992.

[MS] Chiarandini Marco ir Gualandi Stefano. Graph coloring benchmarks. `https://sites.google.com/site/graphcoloring/files`. Žiūrėta: 2020-01-09.

[Mum06] Christine L Mumford. New order-based crossovers for the graph coloring problem. *Parallel problem solving from nature-PPSN IX*, p.p. 880–889. Springer, 2006.

[NC03] Caleb C Noble ir Diane J Cook. Graph-based anomaly detection. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, p.p. 631–636. ACM, 2003.

[OSH87] IM Oliver, DJd Smith ir John RC Holland. Study of permutation crossover operators on the traveling salesman problem. *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*. Hillsdale, NJ: L. Erlhaum Associates, 1987., 1987.

[PS78] Christos H Papadimitriou ir Kenneth Steiglitz. Some examples of difficult traveling salesman problems. *Operations Research*, 26(3):434–443, 1978.

[Rei] Gerhard Reinelt. Library of sample instances for the tsp (and related problems) from various sources and of various types. `http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html`. Žiūrėta: 2020-01-09.

[SBD+08] Jimmy Secretan, Nicholas Beato, David B D Ambrosio, Adelein Rodriguez, Adam Campbell ir Kenneth O Stanley. Picbreeder: evolving pictures collaboratively online. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, p.p. 1759–1768. ACM, 2008.

[SYH09] Yizhou Sun, Yintao Yu ir Jiawei Han. Ranking-based clustering of heterogeneous information networks with star network schema. *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, p.p. 797–806. ACM, 2009.

[SL15] Kenneth O. Stanley ir Joel Lehman. *Why Greatness Cannot Be Planned: The Myth of the Objective*. Springer publisher company, 2015.

[Sta07] Kenneth O Stanley. Compositional pattern producing networks: a novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007.

[TY06] Jun-ichi Takeuchi ir Kenji Yamanishi. A unifying framework for detecting outliers and change points from time series. *IEEE transactions on Knowledge and Data Engineering*, 18(4):482–492, 2006.

[Woo97] David R Wood. An algorithm for finding a maximum clique in a graph. *Operations Research Letters*, 21(5):211–217, 1997.