

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS INSTITUTAS
INFORMATIKOS KATEDRA

Kursinis darbas

Rikiavimo tobulinimas genetiniais algoritmais
(Improving sorting with genetic algorithms)

Atliko: 3 kurso 2 grupės studentas

Deividas Zaleskis (parašas)

Darbo vadovas:

Irmantas Radavičius (parašas)

Vilnius
2021

Turinys

Išvadas	2
1. Šelo rikiavimo algoritmas	5
1.1. Šelo rikiavimo algoritmo efektyvumas.....	5
1.1.1. Šelo rikiavimo algoritmo asimptotinė analizė	5
1.1.2. Šelo rikiavimo algoritmo efektyvumas priklausomai nuo duomenų dydžio	5
1.1.3. Šelo rikiavimo algoritmo efektyvumas priklausomai nuo duomenų specifikos	5
1.2. Šelo rikiavimo algoritmo versijos	6
1.2.1. Vadovėlinis Šelo rikiavimo algoritmas	6
1.2.2. Patobulintas Šelo rikiavimo algoritmas	7
2. Tarpų sekų efektyvumo kriterijų nustatymas	8
3. Tarpų sekų generavimas	9
3.1. Genetiniai algoritmai	9
3.1.1. Chromosomos	9
3.1.2. Tinkamumas	9
3.1.3. Populiacija	9
3.1.4. Genetiniai operatoriai	9
3.1.5. Genetinio algoritmo sustojimo sąlygos.....	10
3.1.6. Galima paprasto genetinio algoritmo implementacija	10
3.2. Trumpų tarpų sekų generavimas	11
3.3. Vidutinio ilgio tarpų sekų generavimas	12
4. Eksperimentų vykdymo aplinkos paruošimas	13
4.0.1. Operacijų skaičiavimas	13
4.0.2. Veikimo laiko matavimas	13
4.0.3. Pradinių duomenų generavimas	13
4.1. Techninės detalės.....	13
5. Tarpų sekų efektyvumo įvertinimas	14
5.1. Pasiruošimas tarpų sekų vertinimui	14
5.2. Trumpų tarpų sekų efektyvumo įvertinimas	14
5.2.1. Trumpų tarpų sekų efektyvumo įvertinimas naudojant vadovėlinį Šelo algoritmą .	15
5.2.2. Trumpų tarpų sekų efektyvumo įvertinimas naudojant patobulintą Šelo algoritmą	16
5.3. Vidutinio ilgio tarpų sekų efektyvumo įvertinimas.....	16
5.3.1. Vidutinio ilgio tarpų sekų efektyvumo įvertinimas naudojant vadovėlinį Šelo al-	
goritmą.....	17
5.3.2. Vidutinio ilgio tarpų sekų efektyvumo įvertinimas naudojant patobulintą Šelo	
algoritmą.....	18
Išvados	20
Literatūra	21

Įvadas

Viena pagrindinių informatikos sąvokų yra algoritmas. Formaliai algoritmą galima apibūdinti kaip baigtinę seką instrukcijų, nurodančių kaip rasti nagrinėjamo uždavinio sprendinį. Algoritmo koncepcija egzistuoja nuo senovės laikų [Knu72], tačiau atsiradus kompiuteriams, tapo įmanoma algoritmų vykdymą automatizuoti, paverčiant juos mašininio kodu suprantamu kompiuteriams [WWG51]. Taip informatikos mokslas nuo teorinių šaknų [Tur37] įgavo ir taikomąją pusę. Beveik visus algoritmus galima suskirstyti į dvi klases: kombinatorinius algoritmus ir skaitinius algoritmus. Skaitiniai algoritmai sprendžia tolydžius uždavinius: optimizuoti realaus argumento funkciją, išspręsti tiesinių lygčių sistemą su realiais koeficientais, etc. Kombinatoriniai algoritmai sprendžia diskrečius uždavinius ir operuoja diskrečiais objektais: skaičiais, sąrašais, grafais, etc. Vienas žinomiausių diskrečiaus uždavinio pavyzdžių yra duomenų rikiavimas.

Duomenų rikiavimas yra vienas pamatinių informatikos uždavinių. Matematiškai jis formuluojamas taip: duotai baigtinei palyginamų elementų sekai $S = (s_1, s_2, \dots, s_n)$ pateikti tokį kėlinį, kad pradinės sekos elementai būtų išdėstyti didėjančia (mažėjančia) tvarka [RB13]. Rikiavimo uždavinys yra aktualus nuo pat kompiuterių atsiradimo ir buvo laikomas vienu pagrindinių diskrečių uždavinių, kuriuos turėtų gebėti spręsti kompiuteris [Knu70]. Rikiavimo uždavinio sprendimas dažnai padeda pagrindą efektyviam kito uždavinio sprendimui, pavyzdžiui, atliekant paiešką sąraše, galima taikyti dvejetainės paieškos algoritmą tik tada, kai sąrašas yra išrikiuotas. Kadangi rikiavimo uždavinys yra fundamentalus, jam spręsti egzistuoja labai skirtingų algoritmų.

Rikiavimo algoritmų yra įvairių: paremtų palyginimu (elementų tvarką nustato naudojant palyginimo operatorius), stabilių (nekeičia lygių elementų tvarkos), nenaudojančių papildomos atminties (atminties sudėtingumas yra $O(1)$), etc. Asimptotiškai optimalūs palyginimu paremti algoritmai blogiausiu atveju turi $O(n \log n)$ laiko sudėtingumą, o ne palyginimu paremti algoritmai gali veikti dar greičiau, tačiau nėra tokie universalūs, kadangi rikiuojama remiantis duomenų specifika. Tiesa, rikiuojant remtis vien algoritmo asimptotika nepakanka: rikiavimas įterpimu (angl. insertion sort) blogiausiu atveju turi $O(n^2)$ laiko sudėtingumą [BFM06], tačiau mažesnius elementų kiekius rikiuoja daug greičiau, nei asimptotiškai optimalūs algoritmai, pavyzdžiui, rikiavimas krūva (angl. heapsort) [For64]. Todėl pastaruoju metu plačiai naudojami hibridiniai rikiavimo algoritmai, kurie sujungia keletą rikiavimo algoritmų į vieną ir panaudoja jų geriausias savybes. Nepaisant įvairovės ir naujų algoritmų gausos, klasikiniai rikiavimo algoritmai išlieka aktualūs.

Šelo rikiavimo algoritmas (angl. Shellsort, toliau - Šelo algoritmas) [She59] yra paremtas palyginimu, nenaudojantis papildomos atminties ir nestabilus. Šelo algoritmą galima laikyti rikiavimo įterpimu modifikacija, kuri lygina ne gretimus, o toliau vienas nuo kito esančius elementus, taip paspartindama jų perkėlimą į galutinę poziciją. Pagrindinė algoritmo idėja - išskaidyti rikiuojamą seką S į posekius S_1, S_2, \dots, S_n , kur kiekvienas posekis $S_i = (s_i, s_{i+h}, s_{i+2h}, \dots)$ yra sekos S elementai, kurių pozicija skiriasi h . Išrikiavus visus sekos S posekius S_i su tarpu h , seka tampa h -išrikiuota. Remiantis tuo, jog sekai S esant h -išrikiuota ir ją k -išrikiavus, ji lieka h -išrikiuota [GK72], galima kiekvieną algoritmo iteraciją mažinti tarpą, taip vis didinant sekos S išrikiuotumą.

Pritaikant šias idėjas ir rikiavimui naudojant mažėjančią tarpų seką su paskutiniu nariu 1, kuris garantuoja rikiavimą įterpimu paskutinėje iteracijoje, galima užtikrinti, jog algoritmo darbo pabaigoje seka S bus pilnai išrikiuota. Įvertinant Šelo algoritmo idėjas, nesunku pastebėti tarpų sekų įtaką jo veikimui.

Šelo algoritmo efektyvumas tiesiogiai priklauso nuo pasirinktos tarpų sekos. Weiss atlikto tyrimo [Wei91] rezultatai rodo, jog su Sedgewick pasiūlyta seka šis algoritmas veikia beveik dvigubai greičiau nei Šelo pradinis variantas, kai $n = 1000000$. Yra įrodyta, kad Šelo algoritmo laiko sudėtingumo blogiausiu atveju apatinė riba yra $\Omega(\frac{n \log^2 n}{\log \log n^2})$ [PPS92], taigi jis nėra asimptotiškai optimalus. Tiesa, kol kas nėra rasta seka, su kuria Šelo algoritmas pasiektų šią apatinę ribą. Kiek žinoma autoriui, asimptotiškai geriausia tarpų seka yra rasta Pratt, kuri yra formos $2^p 3^p$ ir turi $\Theta(n \log^2 n)$ laiko sudėtingumą [Pra72], tačiau praktikoje ji veikia lėčiau už Ciura [Ciu01] ar Tokuda [Tok92] pasiūlytas sekas. Daugelio praktikoje efektyvių sekų asimptotinis sudėtingumas laiko atžvilgiu lieka atvira problema, nes jos yra rastos eksperimentiškai. Vienas iš metodų, kuriuos galima taikyti efektyvių tarpų sekų radimui, yra genetinis algoritmas.

Genetinis algoritmas (GA) yra metodas rasti euristicas, paremtas biologijos žiniomis apie natūralios atrankos procesą. Kartu su genetiniu programavimu, evoliuciniais algoritmais ir kitais metodais, genetiniai algoritmai sudaro evoliucinių skaičiavimų šeimą. Visi šios šeimos atstovai yra paremti pradinės populiacijos generavimu ir iteraciniu populiacijos atnaujinimu naudojant biologijos įkvėptas strategijas. J.H. Holland, GA pradininkas, savo knygoje [Hol92] apibrėžė genetinio algoritmo sąvoką ir su ja glaudžiai susijusias chromosomų (potencialių uždavinio sprendinių, išreikštų genų rinkiniu), bei rekombinacijos (tėvinių chromosomų genų perdavimo palikuonims), atrankos (tinkamiausių chromosomų atrinkimo) ir mutacijos (savaiminio chromosomos genų kitimo) operatorių koncepcijas. Genetinių algoritmų veikimo strategija pagrįsta pradinės chromosomų populiacijos evoliucija, kiekvienos naujos chromosomų kartos gavimui naudojant rekombinacijos, atrankos ir mutacijos operatorius. Toliau bus aptariamos genetinių algoritmų taikymo galimybės.

Genetiniai algoritmai taikomi sprendžiant įvairius paieškos ir optimizavimo uždavinius, kuomet nesunku nustatyti, ar sprendinys tinkamas, tačiau tinkamo sprendinio radimas reikalauja daug resursų ar net pilno perrinkimo. Tokiu atveju apytikslio sprendinio radimas (euristika) gali būti daug patrauklesnis sprendimo būdas, kadangi tikslaus sprendinio radimas dažnai yra NP-sunkus uždavinys. Todėl GA yra pritaikomi sudarant grafikus ir tvarkaraščius, sprendžiant globalaus optimizavimo uždavinius ir net projektuojant NASA mikrosatelitų antenas [HGL⁺06]. Nesunku pastebėti, jog efektyvių Šelo algoritmo tarpų sekų radimas yra sunkus uždavinys atliekamų skaičiavimų prasme, tikėtina reikalaujantis pilno potencialių sprendinių perrinkimo, tad šio uždavinio sprendimui taikyti GA yra prasminga. Kiek žinoma autoriui, kol kas yra buvę du bandymai taikyti genetinius algoritmus efektyvių Šelo algoritmo tarpų sekų radimui [RBH⁺02; SY99]. Abiejuose darbuose teigiama, jog genetiniais algoritmais gautos tarpų sekos veikia greičiau už Sedgewick seką, kuri literatūroje laikoma viena efektyviausių.

Darbo **tikslas**: pritaikyti genetinius algoritmus Šelo algoritmo tarpų sekoms generuoti.

Darbo uždaviniai:

- Atlikti Šelo rikiavimo algoritmo literatūros analizę.
- Nustatyti kriterijus tarpų sekų efektyvumui įvertinti.
- Naudojant genetinius algoritmus sugeneruoti tarpų sekas.
- Paruošti aplinką eksperimentų vykdymui.
- Atliekant eksperimentus įvertinti sugeneruotų ir pateiktų literatūroje tarpų sekų efektyvumą.

Šis darbas sudarytas iš 5 skyrių. Pirmame skyriuje atliekama Šelo rikiavimo algoritmo literatūros analizė. Antrame skyriuje nustatomi kriterijai tarpų sekų efektyvumui įvertinti. Ketvirtame skyriuje naudojant genetinius algoritmus generuojamos tarpų sekos. Penktame skyriuje paruošiama eksperimentų vykdymo aplinka. Šeštame skyriuje atliekant eksperimentus įvertinamas sugeneruotų ir pateiktų literatūroje tarpų sekų efektyvumas.

1. Šelo rikiavimo algoritmas

1.1. Šelo rikiavimo algoritmo efektyvumas

1.1.1. Šelo rikiavimo algoritmo asimptotinė analizė

1.1.2. Šelo rikiavimo algoritmo efektyvumas priklausomai nuo duomenų dydžio

Renkantis kokį algoritmą naudoti, labai svarbu įvertinti tikėtiną duomenų dydį. Šelo rikiavimo algoritmas yra efektyviausias, kai duomenų dydis yra ganėtinai mažas [Ciu01]. Kaip teigiama [SY99], šis algoritmas veikia geriausiai, kai $N \leq 106$. Tokiu atveju, jis lenkia net ir vienu greičiausių laikomą greitojo rikiavimo algoritmą. Todėl Šelo rikiavimo algoritmas dažnai naudojamas hibridiniuose rikiavimo algoritmuose, kai pasiekus tam tikrą rekursijos lygį rikiuojamų duomenų dalis tampa pakankamai maža ir tolimesnė rekursija asimptotiškai optimaliu algoritmu nebeturi prasmės, kadangi jos tęsimas reikalautų per daug mašinos resursų. Tokių šio algoritmo naudojimo pavyzdžių galima rasti Go programavimo kalbos standartinėje bibliotekoje [Aut09] bei bzip2 failų glaudinimo programoje [Sew10]. Tiesa, net ir rikiuojant didesnius duomenų dydžius, Šelo algoritmas nėra labai lėtas [Ciu01]. Todėl jis yra vertingas įrankis įgyvendinant operacinės sistemos branduolį ar programuojant įterptinėms sistemoms, kadangi dėl ribotų atminties išteklių asimptotiškai optimalūs algoritmai, kurie dažniausia yra rekursyvūs ir naudoja daugiau atminties, tokiais atvejais netinka. Taigi, nors Šelo rikiavimo algoritmas nėra vienas greičiausių kai uždavinys didelis, yra scenarijų, kuriais šį algoritmą rinktis tikrai verta.

1.1.3. Šelo rikiavimo algoritmo efektyvumas priklausomai nuo duomenų specifikos

Viena palankiausių Šelo algoritmo savybių yra jo adaptyvumas. Adaptyvumas rikiavimo algoritmų kontekste reiškia, jog algoritmas atlieka mažiau operacijų, jei duomenys dalinai išrikiuoti. Adaptyvumą Šelo algoritmas paveldi iš rikiavimo įterpimu, nes yra jo optimizacija. Verta pastebėti, jog Šelo algoritmas būtent šia savybe ir remiasi rikiavimo įterpimu optimizavimui: kadangi rikiavimas įterpimu veikia labai greitai, kai duomenys išrikiuoti (tokiu atveju sudėtingumas laiko atžvilgiu yra $O(n)$), duomenis pradžioje h-išrikiavus, paskutine iteracija taikomas rikiavimas įterpimu veikia daug greičiau. Moksliniuose tyrimuose ši savybė retai turi įtakos, kadangi dažniausia siekiama ištirti algoritmo veikimą, kai duomenys nėra tvarkingai išdėstyti ir tipiškai renkamosi atsitiktinai generuoti pradiniai duomenis šiam tikslui pasiekti. Savaimė aišku, jog praktikoje retai pavyks sutikti visiškai atsitiktinai išsidėsčiusius duomenis, neturinčius nei vieno išrikiuoto posekio ilgesnio nei 1. Kaip parodo [CK80], Šelo algoritmas veikia net keletą kartų greičiau, kai duomenys yra pilnai išrikiuoti ir veikimo sparta nusileidžia tik rikiavimui įterpimu su visais duomenų dydžiais. Kai duomenų dydis nedidelis (iki 200 elementų) ir 20% elementų nėra teisingose pozicijose, Šelo algoritmas veikimo sparta nusileidžia tik greitojo rikiavimo algoritmui. Palankūs rezultatai pateikiami ir [FG15] - su dalinai išrikiuotais duomenimis Šelo algoritmas veikia apytiksliai 1.5 karto greičiau, o su išrikiuotais duomenimis pasiekiamas apie 3.8 karto mažesnis veikimo greitis. Taigi,

rikiuojant bent dalinai išrikiuotus duomenis Šelo algoritmas yra labai efektyvus.

1.2. Šelo rikiavimo algoritmo versijos

1.2.1. Vadovėlinis Šelo rikiavimo algoritmas

Kaip ir daugelis algoritmų, Šelo rikiavimo algoritmas turi keletą galimų implementacijų. Žinomiausia iš jų, be abejo, yra vadovėlinė šio algoritmo versija (toliau - VŠA), daugeliui pasaulio informatikos studentų žinoma iš duomenų struktūrų ir algoritmų kurso. Jos pseudokodas pateikiamas žemiau.

Algorithm 1 Vadovėlinis Šelo rikiavimo algoritmas

```
1: foreach  $gap$  in  $H$  do
2:   for  $i \leftarrow gap + 1$  to  $N$  do
3:      $j \leftarrow i$ 
4:      $temp \leftarrow S[i]$ 
5:     while  $j > gap$  and  $S[j - gap] > S[j]$  do
6:        $S[j] \leftarrow S[j - gap]$ 
7:        $j \leftarrow j - gap$ 
8:     end while
9:      $S[j] \leftarrow temp$ 
10:   end for
11: end for
```

1.2.2. Patobulintas Šelo rikiavimo algoritmas

Verta pastebėti, jog vadovėlinė Šelo algoritmo implementacija nėra naši [RB13]. Vykdamas vidinį vadovėlinio Šelo algoritmo ciklą (1.5 - 1.8 eilutės), 1.5 eilutėje yra tikrinama, ar $S[j]$ jau yra tinkamoje pozicijoje. Jei $S[j]$ jau yra tinkamoje pozicijoje, vidinis ciklas nėra vykdomas ir jokių elementų pozicijos nėra keičiamos. Tačiau 1.4 ir 1.9 eilutėse vis tiek yra vykdomi du priskyrimai, kurie šiuo atveju nueina veltui (ir tai nutinka pakankamai dažnai). Siekiant tai ištaisyti, [RB13] pateikė patobulintą Šelo algoritmo versiją (toliau - PŠA), kuri prieš vykdydama bet kokius kitus žingsnius patikrina, ar elementas $S[j]$ jau yra tinkamoje pozicijoje. Patobulinto Šelo algoritmo pseudokodas pateikiamas žemiau.

Algorithm 2 Patobulintas Šelo rikiavimo algoritmas

```
1: foreach  $gap$  in  $H$  do
2:   for  $i \leftarrow gap + 1$  to  $N$  do
3:     if  $S[i - gap] > S[i]$  then
4:        $j \leftarrow i$ 
5:        $temp \leftarrow S[i]$ 
6:       repeat
7:          $S[j] \leftarrow S[j - gap]$ 
8:          $j \leftarrow j - gap$ 
9:       until  $j \leq gap$  or  $S[j - gap] \leq S[j]$ 
10:       $S[j] \leftarrow temp$ 
11:     end if
12:   end for
13: end for
```

2. Tarpų sekų efektyvumo kriterijų nustatymas

Šelo algoritmo tarpų sekų efektyvumo įvertinimas nėra trivialus. Rikiavimo algoritmai dažniausiai yra vertinami pagal atliekamų priskyrimų skaičių, kadangi daugelyje algoritmų priskyrimų skaičius uždaviniui augant greitai artėja prie palyginimų skaičiaus ir tokiu metodu gautas įvertis būna pakankamai tikslus. Tačiau Šelo algoritmo atveju, atliekamų palyginimų ir priskyrimų skaičiaus santykis augant N nebūtinai artėja prie 1 [RB13]. Taigi, vien priskyrimų skaičius nėra pakankamai tikslus kriterijus tarpų sekų efektyvumui įvertinti, kadangi juo remiantis gautas įvertis pilnai neatspindi praktinio sekos efektyvumo. Kaip parodo [Ciu01], šiame algoritme dominuojanti operacija yra palyginimas ir efektyvios sekos turėtų jų atlikti kuo įmanoma mažiau. Apibendrinant, atliekamų palyginimų skaičius yra tinkamesnis kriterijus Šelo algoritmo tarpų sekų efektyvumui įvertinti.

Matuojant rikiavimo algoritmo efektyvumą tik naudojant sveikaskaitinius pradinius duomenis, gauti rezultatai gali būti netikslūs, kadangi palyginimo ir priskyrimo operacijų sparta priklauso nuo rikiuojamų duomenų tipo. Rikiuojant simbolių eilutes, priskyrimas atliekamas naudojant rodykles, kas yra $O(1)$ operacija, tačiau palyginimas yra $O(n)$ blogiausiu atveju. Ir atvirkščiai, rikiuojant įrašus, priskyrimas reikalauja perkopijuoti visą įrašą, tad šios operacijos sudėtingumas yra $O(n)$, o palyginimas gali būti atliekamas naudojant tam tikrą raktą, taigi tik $O(1)$. Todėl apsiriboti vien palyginimų ar priskyrimų skaičiumi nepakanka, kadangi tiksliausia praktinio algoritmo veikimo laiko aproksimacija (tai, kam ir skaičiuojamos operacijos), bus gauta tik įvertinant abu šiuos rodiklius.

Algoritmo veikimo laikas, nors ir priklausomas nuo platformos, kurioje vykdomas tyrimas, detalių, taip pat gali duoti tinkamų įžvalgų įvertinant praktinį efektyvumą. Kadangi algoritmo atliekamos operacijos skaičiuojamos tam, jog gauti veikimo laiko aproksimaciją, tai realus veikimo laikas yra konkretus įvertis, leidžiantis praktiškai įvertinti duotos sekos efektyvumą. Taigi, įvertinant tarpų sekų efektyvumą bus remiamasi visomis atliekamomis operacijomis bei realiais veikimo laikais.

3. Tarpų sekų generavimas

3.1. Genetiniai algoritmai

Prieš pradedant generuoti sekas, reikalinga plačiau apžvelgti genetinio algoritmo sąvoką. Paprasčiausias genetinis algoritmas susideda iš chromosomų populiacijos bei atrankos, mutacijos ir rekombinacijos operatorių.

3.1.1. Chromosomos

Projektuojant genetinį algoritmą tam tikro uždavinio sprendimui, svarbu tinkamai pasirinkti, kaip kompiuteriu modeliuoti galimus sprendinius (chromosomas). Chromosomos kompiuterio atmintyje standartiškai išreiškiamos bitų eilutėmis, kadangi tai palengvina tiek mutaciją (pakanka apversti kurio nors atsitiktinio bito reikšmę), tiek rekombinaciją (pakanka perkopijuoti pasirinktus tėvinių chromosomų bitus į vaikinę chromosomą). Tiesa, tai nėra vienintelis įmanomas būdas, ir kai kurių uždavinių sprendiniai modeliuojami pvz. grafu ar simbolių eilute.

3.1.2. Tinkamumas

Sprendinio kokybę įvardijame kaip jo tinkamumą, kuris apibrėžiamas tinkamumo funkcijos reikšme, pateikus sprendinį kaip parametą. Nesunku pastebėti, jog tinkamumo funkcija tėra tikslo funkcijos specializacija, kuri naudojama chromosomų vertinimui. Tinkamumo funkcija yra viena svarbiausių genetinio algoritmo dalių, kadangi kai ji netinkamai parinkta, algoritmas nekonverguos į tinkamą sprendinį arba užtruks labai ilgai.

3.1.3. Populiacija

Chromosomų rinkinys, literatūroje dažnai vadinamas populiacija, atspindi uždavinio sprendinių aibę, kuri kinta kiekvieną genetinio algoritmo iteraciją. Jei GA suprojektuotas tinkamai, dažnu atveju vidutinis populiacijos tinkamumas gerės kiekvieną iteraciją [SY99].

3.1.4. Genetiniai operatoriai

Genetinis algoritmas vykdymo metu iteratyviai atnaujiną esamą populiaciją, kurdamas naujas kartas taikant atrankos, rekombinacijos ir mutacijos operatorius. Atrankos operatorius grąžina tinkamiausius populiacijos individus, kuriems yra leidžiama susilaukti palikuonių taikant rekombinacijos operatorių. Rekombinacijos operatorius veikia iš dviejų tėvinių chromosomų sukurdamas naują vaikinę chromosomą, kas dažniausiai pasiekama tam tikru būdu perkopijuojant tėvų genų atkarpas į vaikinę chromosomą. Rekombinacijos strategijų yra įvairių, tačiau ne kiekvienam uždaviniui visos jos tinka, kadangi kai kuriais atvejais netinkamai parinkta rekombinacijos strategija pagamina neatitinkančią uždavinio apribojimų vaikinę chromosomą. Pavyzdžiui, jei modeliuojama

chromosoma yra sąrašas, turintis susidėti iš tam tikrų elementų, neįmanoma garantuoti, jog atsitiktinai perkopijavus tėvinių chromosomų genus į vaikinę chromosomą šis apribojimas bus išlaikytas. Galiausiai tam tikrai populiacijos daliai yra pritaikomas mutacijos operatorius. Jo veikimo principas yra gana paprastas: pasirinktos chromosomos vienas ar keli genai yra modifikuojami, nežymiai pakeičiant jų reikšmes ar sukeičiant kelių genų reikšmes vietomis. Mutacijos operatorius praplečia vykdomos paieškos erdvę, o tai labai svarbu, kadangi kitaip algoritmas gali konverguoti į lokalinio minimumo taškus, taip ir nepasiekdamas globaliojo minimumo. Atsižvelgiant į tai, yra laikoma, jog mutacijos operatorius yra kertinė genetinio algoritmo dalis, kuri palaiko genetinę individų įvairovę ir padeda rasti tinkamiausius sprendinius.

3.1.5. Genetinio algoritmo sustojimo sąlygos

Kad genetinis algoritmas neveiktų amžinai, yra pasirenkama viena ar kelios sustojimo sąlygos, pvz. pasiektas maksimalus kartų skaičius, vidutinio tinkamumo pokytis tarp paskutinių dviejų populiacijos kartų pakankamai mažas. Kiekvieną iteraciją GA patikrina, ar kažkuri iš sustojimo sąlygų yra patenkinta ir jei taip - baigia darbą.

3.1.6. Galima paprasto genetinio algoritmo implementacija

Apjungiant visas aukščiau aptartas genetinio algoritmo dalis, galima suformuluoti paprasto genetinio algoritmo pseudokodą.

Algorithm 3 Paprastas GA

Let *current_population* be a random population of chromosomes.

Let *mutation_rate* be a real value between 0 and 1.

Let *crossover_fraction* be a real value between 0 and 1.

repeat

 Apply the fitness function to each chromosome in *current_population*.

 Sort the chromosomes in *current_population* based on their fitness.

 Let *new_population* be an empty set.

repeat

 Let (*parent1*, *parent2*) be a pair of parent chromosomes selected randomly from the first $[N * \text{crossover_fraction}]$ elements of *current_population*.

 Let *child_chromosome* be the result of applying the crossover operator to *parent1* and *parent2*.

if $\text{rand}() < \text{mutation_rate}$ **then**

 Apply the mutation operator to *child_chromosome*

end if

 Add *child_chromosome* to *new_population*

until N offspring have been created.

 Assign *new_population* to *current_population*.

until Termination criteria are satisfied.

3.2. Trumpų tarpų sekų generavimas

Pirmame etape buvo generuojamos tarpų sekos, kurios efektyvios kai $N = 1000$. Trumpų sekų generavimui buvo paruoštas vienkriterinis genetinis algoritmas. GA implementacijai buvo pasirinkta OpenGA biblioteka [MAM⁺17] dėl suteikiamos laisvės pasirinkti, kaip įgyvendinti genetinius operatorius bei modernių kalbos konstruktų ir lygiagretaus vykdymo palaikymo. Remiantis [SY99], trumpos sekos chromosoma buvo modeliuojama kaip septynių sveikų skaičių masyvas. Chromosomos buvo vertinamos jas naudojant 20-ties atsitiktinai sugeneruotų sveikų skaičių masyvų rikiavimui vadovėline Šelo rikiavimo algoritmo versija ir skaičiuojant atliktas palyginimo operacijas. Chromosomos tinkamumo funkcija buvo apibrėžta kaip atliktų palyginimų skaičiaus aritmetinis vidurkis. Rekombinacijos operatorius buvo įgyvendintas tolygia strategija, kur abiejų tėvų genai turi vienodą tikimybę būti perduoti vaicinei chromosomai. GA sustojimo kriterijumi buvo pasirinkta laikyti tinkamiausio individo nepakitimą 25 iteracijas. Mutacijos operatorius buvo įgyvendintas su $\frac{1}{5}$ tikimybe keičiant tam tikrą chromosomą, pridėdant prie atsitiktinai pasirinkto geno reikšmės skaičių $(\text{rand}() - \text{rand}()) * \frac{25}{\sqrt{\text{karta}}}$ su apribojimu, jog pakeistas genas turi priklausyti intervalui $[1, 1000]$. Siekiant išvengti netinkamų sprendinių, kiekviena seka po rekombinacijos ar mutacijos operatorių taikymo buvo išrikiuojama ir užtikrinama, jog paskutinis sekos narys yra 1.

GA buvo vykdomas su atsitiktinai sugeneruota 10000 individų populiacija, kiekvieną iteraciją pritaikant rekombinacijos operatorių $\frac{1}{2}$ populiacijos. Tokiu būdu buvo sugeneruota ir atrinkta 10 sekų. Po to iš visų sugeneruotų sekų buvo atrinkta tinkamiausia, sugeneruotas sekas naudojant 10000 atsitiktinai sugeneruotų masyvų ilgio 1000 rikiavimui ir matuojant atliktų palyginimo operacijų aritmetinį vidurkį. Atlikus matavimus, buvo pasirinkta seka 855, 264, 86, 35, 12, 5, 1.

3.3. Vidutinio ilgio tarpų sekų generavimas

Antrame etape buvo generuojamos tarpų sekos, kurios efektyvios kai $N = 100000$. Vidutinio ilgio sekų generavimui buvo pasitelktas modifikuotas genetinis algoritmas, naudotas generuojant trumpas tarpų sekas. Tarpų sekos chromosoma buvo modeliuojama kaip sveikų skaičių masyvas ilgio 12. Buvo atsisakyta chromosomų vertinimo naudojant 20 skirtingų masyvų, kadangi tai darė neigiamą įtaką genetinio algoritmo veikimo laikui. Vietoje to, kiekvienos sekos vertinimui buvo naudojami 5 atsitiktinai sugeneruoti masyvai ilgio 100000. GA buvo nurodyta sustoti, kai kartų skaičius pasieks 100. Mutacijos operatorius buvo pakoreguotas ir su $\frac{1}{5}$ tikimybe keitė tam tikrą chromosomą, pridėdam prie atsitiktinai pasirinkto geno reikšmės skaičių $(rand() - rand()) * \frac{75}{\sqrt{karta}}$.

GA buvo vykdomas su atsitiktinai sugeneruota 1000 individų populiacija, paskutinius septynis chromosomos genus inicializuojant elementais 855, 264, 86, 35, 12, 5, 1, t.y. sekos, gautos pirmame etape, nariais. Siekiant išlaikyti tinkamiausius sprendinius, buvo pasitelktas elitizmas - geriausiai pasirodžiusios 75 chromosomos nekeistos patekdavo į kitą algoritmo iteraciją. Rekombinacijos operatorius buvo taikomas $\frac{1}{2}$ populiacijos kiekvieną algoritmo iteraciją. Tokiu būdu buvo sugeneruota ir atrinkta 10 sekų. Po to iš visų sugeneruotų sekų buvo atrinkta tinkamiausia, naudojant tokį pat metodą kaip ir praeitame etape, tik pakeitus rikiuojamo masyvo dydį į 100000. Atlikus matavimus, buvo pasirinkta seka 45794, 17396, 7414, 3136, 1206, 561, 264, 86, 35, 12, 5, 1.

4. Eksperimentų vykdymo aplinkos paruošimas

4.0.1. Operacijų skaičiavimas

Siekiant išmatuoti VŠA bei PŠA atliekamų operacijų skaičių, buvo parengtos modifikuotos šių algoritmų versijos, kurios kaip rezultatą grąžina atliktus palyginimus bei priskyrimus.

4.0.2. Veikimo laiko matavimas

Verta pastebėti, jog matuoti veikimo laiką naudojant operacijas skaičiuojančius algoritmus nėra tinkama, kadangi pats operacijų skaičiavimas reikalauja papildomų žingsnių, o tai gali iškreipti gautus rezultatus. Atsižvelgiant į tai, veikimo laiko matavimui naudoti nemodifikuoti VŠA ir PŠA algoritmai.

4.0.3. Pradinių duomenų generavimas

Vykdant eksperimentus buvo pasirinkta naudoti sveikaskaitinius pradinius duomenis, kadangi jų generavimas yra pakankamai paprastas. Tai buvo įgyvendinta naudojant MT19937 pseudoatsiktinių skaičių generatorių, inicializuotą sisteminio laiko reikšme. Generuojami duomenys buvo tolygiai paskirstyti nuo INT_MIN iki INT_MAX, kas leido žymiai sumažinti duomenų duplikacijos tikimybę.

4.1. Techninės detalės

Eksperimentų vykdymui buvo naudotas kompiuteris su 2.70 GHz Intel(R) Core(TM) i7-10850H procesoriumi, 32 GB operatyviosios atminties ir Windows 10 operacine sistema. Tyrimas buvo įgyvendintas C++ kalba su GNU g++ 8.1.0 kompiliatoriumi naudojant -O3 optimizacijos lygį.

5. Tarpų sekų efektyvumo įvertinimas

5.1. Pasiruošimas tarpų sekų vertinimui

Iš anksto verta pastebėti, jog ne visos šiame skyriuje tiriamos tarpų sekos iš tiesų yra būtent tokio ilgio, koks pateikiamas šiame darbe. Tačiau būtina atsižvelgti į faktą, jog tai galėtų sukelti nelygias sąlygas atliekamuose matavimuose, kadangi seka, turinti elementų didesnių už maksimalų tiriamą N , atliktų nereikalingas operacijas, o per trumpa seka veiktų neefektyviai. Siekiant standartizuoti tarpų sekų ilgį, buvo pasirinkta kai kurias sekas sutrumpinti (ar pratęsti, jei tam yra žinoma rekursyvi formulė) taip, jog standartizuota seka būtų ilgiausia įmanoma seka, kurios visi elementai mažesni už maksimalų tiriamą N . Atliekamų matavimų skaičius buvo pasirinktas atsižvelgiant į tame etape tiriamo N dydį: jei N labai mažas, egzistuoja didelė tikimybė jog gautas vidutinis įvertis bus netikslus, matuojant, pavyzdžiui, tik 1000 kartų. Todėl tiriant trumpesnes sekas buvo stengtasi atlikti daugiau matavimų.

5.2. Trumpų tarpų sekų efektyvumo įvertinimas

Trumpų tarpų sekų efektyvumo tyrimui buvo pasirinktos šios tarpų sekos:

- Tokuda: 525,233,103,46,20,9,4,1
- Ciura: 701,301,132,57,23,10,4,1
- Simpson-Yachavaram: 893,219,83,36,13,4,1
- Sedgewick: 929,505,209,109,41,19,5,1
- Incerpi-Sedgewick: 861,336,112,48,21,7,3,1
- Seka 1: 855,264,86,35,12,5,1

Tokuda [Tok92] seka buvo pasirinkta atsižvelgiant į tai, jog praktikoje ji laikoma viena efektyviausių. Kiek žinoma autoriui, kai $N = 1000$ Ciura [Ciu01] seka atlieka mažiausiai palyginimų vidutiniu atveju, tad jos įtraukimas būtinas. Simpson ir Yachavaram [SY99] seka buvo pasirinkta, nes taip pat buvo sugeneruota naudojant GA ir pasak autorių, tai geriausiai veikianti jų rasta seka, kai $N = 1000$. Sedgewick [Sed86] ir Incerpi-Sedgewick [IS85] sekos pasirinktos dėl tvirto matematinio pagrindo (įrodytas $O(n^{\frac{4}{3}})$ laiko sudėtingumas blogiausiu atveju) bei fakto, jog ilgą laiką šios sekos buvo laikomos vienomomis efektyviausių.

Atsižvelgiant į santykinai nedidelį duomenų dydį, matavimai buvo atlikti 50000 kartų. Gauti duomenys pateikiami suapvalinus iki šimtosios dalies, veikimo laikai pateikiami mikrosekundėmis.

5.2.1. Trumpų tarpų sekų efektyvumo įvertinimas naudojant vadovėlinį Šelo algoritmą

1 lentelė. Vidutinis vadovėlinio Šelo algoritmo atliekamų palyginimų skaičius naudojant trupas tarpų sekas

N	Tokuda	Ciura	S/Y	Sedgewick	I/S	Seka 1
50	289.90	287.76	287.70	293.41	296.37	289.21
100	735.82	733.35	731.49	752.13	759.57	731.87
200	1812.59	1797.92	1798.5	1840.15	1870.94	1793.19
400	4315.6	4276.38	4284.94	4405.07	4464.59	4273.84
800	10061.3	9946.54	9987.18	10260.5	10528.2	9999.9
1000	13128.9	13044.4	13075.4	13395.4	13807.4	13054.4

2 lentelė. Vidutinis vadovėlinio Šelo algoritmo atliekamų priskyrimų skaičius naudojant trupas tarpų sekas

N	Tokuda	Ciura	S/Y	Sedgewick	I/S	Seka 1
50	482.37	473.17	453.48	447.69	487.44	456.58
100	1204.5	1186.33	1137.21	1130.61	1227.55	1135.35
200	2933.02	2862.17	2752.73	2761.21	2977.03	2745.81
400	6898.23	6733.29	6522.93	6629.27	6903.13	6444.15
800	15899.2	15487.9	14758.6	15354.8	15950	14745.8
1000	20707.8	20273.4	19200.7	20089.7	20841.4	19193.6

3 lentelė. Vidutinis vadovėlinio Šelo algoritmo veikimo laikas naudojant trupas tarpų sekas

N	Tokuda	Ciura	S/Y	Sedgewick	I/S	Seka 1
50	1.00	1.02	1.10	1.01	1.14	0.90
100	3.04	2.87	2.75	2.95	2.74	3.09
200	6.35	6.25	6.32	6.00	7.20	6.37
400	16.39	15.06	14.29	14.20	14.77	14.92
800	39.93	37.60	35.41	38.60	43.22	33.30
1000	45.74	44.30	42.07	42.58	44.44	44.10

5.2.2. Trumpų tarpų sekų efektyvumo įvertinimas naudojant patobulintą Šelo algoritmą

4 lentelė. Vidutinis patobulinto Šelo algoritmo atliekamų palyginimų skaičius naudojant trupas tarpų sekas

N	Tokuda	Ciura	S/Y	Sedgewick	I/S	Seka 1
50	289.85	287.85	287.83	293.70	296.33	289.09
100	736.17	733.32	731.70	751.94	759.47	732.01
200	1812.48	1798.04	1798.66	1840.66	1871.29	1793.03
400	4315.89	4276.20	4284.83	4405.79	4464.40	4273.78
800	10061.00	9945.74	9986.13	10259.80	10525.90	9999.26
1000	13130.80	13044.10	13077.20	13393.00	13807.50	13053.10

5 lentelė. Vidutinis patobulinto Šelo algoritmo atliekamų priskyrimų skaičius naudojant trupas tarpų sekas

N	Tokuda	Ciura	S/Y	Sedgewick	I/S	Seka 1
50	312.13	319.66	322.80	335.37	322.30	323.76
100	791.06	797.73	811.35	853.82	822.16	814.18
200	1931.99	1919.31	1999.55	2061.03	1998.26	1996.21
400	4537.96	4545.12	4731.20	4855.45	4767.84	4720.75
800	10502.10	10587.70	11134.90	11121.30	11396.80	11222.50
1000	13866.10	13819.40	14510.70	14523.70	14868.70	14490.70

6 lentelė. Vidutinis patobulinto Šelo algoritmo veikimo laikas naudojant trupas tarpų sekas

N	Tokuda	Ciura	S/Y	Sedgewick	I/S	Seka 1
50	1.30	1.26	1.02	1.32	1.32	1.14
100	3.16	3.16	3.14	3.02	3.36	3.22
200	7.54	7.95	7.54	7.61	8.11	8.00
400	18.52	19.00	17.67	17.29	18.92	18.18
800	42.96	42.60	41.81	40.67	42.23	41.19
1000	56.69	56.88	53.39	53.45	56.38	52.84

5.3. Vidutinio ilgio tarpų sekų efektyvumo įvertinimas

Vidutinio ilgio tarpų sekų efektyvumo tyrimui buvo pasirinktos šios tarpų sekos:

- Tokuda: 68178,30301,13467,5985,2660,1182,525,233,103,46,20,9,4,1
- Ciura: 90927,40412,17961,7983,3548,1577,701,301,132,57,23,10,4,1

- Simpson-Yachavaram: 38291,22927,8992,3568,1488,893,219,83,36,13,4,1
- Incerpi-Sedgewick: 86961,33936,13776,4592,1968,861,336,112,48,21,7,3,1
- Sedgewick: 64769,36289,16001,8929,3905,2161,929,505,209,109,41,19,5,1
- Roos-Bennet-Hannon-Zehner: 91433,72985,13229,5267,2585,877,155,149,131,23,8,1
- Seka 2: 45794,17396,7414,3136,1206,561,264,86,35,12,5,1

Atsižvelgiant į duomenų dydį, matavimai buvo atlikti 10000 kartų. Gauti duomenys pateikiami suapvalinus iki sveikosios dalies, veikimo laikai pateikiami mikrosekundėmis.

5.3.1. Vidutinio ilgio tarpų sekų efektyvumo įvertinimas naudojant vadovėlinį Šelo algoritmą

7 lentelė. Vidutinis vadovėlinio Šelo algoritmo atliekamų palyginimų skaičius naudojant vidutinio ilgio tarpų sekas

N	Tokuda	Ciura	S/Y	I/S	Sedgewick	R/B/H/Z	Seka 2
5000	86930	86490	88075	93869	89056	102065	87131
10000	192625	191703	195221	209514	197067	225459	193092
20000	423598	421301	429228	465748	433320	493530	423824
40000	922922	918490	936240	1032259	943633	1070908	923609
80000	1999095	1992131	2033327	2273358	2045446	2317061	2002290
100000	2564038	2552126	2605061	2923557	2623763	2959526	2563334

8 lentelė. Vidutinis vadovėlinio Šelo algoritmo atliekamų priskyrimų skaičius naudojant vidutinio ilgio tarpų sekas

N	Tokuda	Ciura	S/Y	I/S	Sedgewick	R/B/H/Z	Seka 2
5000	134747	132482	129043	138209	133685	140459	129181
10000	296787	292056	284474	306461	295196	311298	284895
20000	648807	638480	623546	672463	650675	680242	622844
40000	1407318	1385991	1358871	1473610	1414589	1466640	1352658
80000	3034012	3001389	2959087	3214096	3072581	3127946	2925254
100000	3888404	3836014	3777087	4124557	3941499	4007456	3735445

9 lentelė. Vidutinis vadovėlinio Šelo algoritmo veikimo laikas naudojant vidutinio ilgio tarpų sekas

N	Tokuda	Ciura	S/Y	I/S	Sedgewick	R/B/H/Z	Seka 2
5000	312	310	294	330	311	296	314
10000	707	702	671	690	695	618	688
20000	1611	1550	1515	1577	1484	1345	1477
40000	4207	4274	4067	4273	4239	3728	4135
80000	9638	9363	8541	9064	9132	8058	8745
100000	13810	12058	11077	11652	11976	9225	9314

5.3.2. Vidutinio ilgio tarpų sekų efektyvumo įvertinimas naudojant patobulintą Šelo algoritmą

10 lentelė. Vidutinis patobulinto Šelo algoritmo atliekamų palyginimų skaičius naudojant vidutinio ilgio tarpų sekas

N	Tokuda	Ciura	S/Y	I/S	Sedgewick	R/B/H/Z	Seka 2
5000	86926	86499	88071	93852	89060	102054	87125
10000	192624	191711	195229	209574	197045	225524	193106
20000	423593	421326	429232	465791	433302	493515	423825
40000	922899	918450	936191	1032317	943633	1071022	923625
80000	1998794	1992412	2033313	2272227	2045555	2317464	2002261
100000	2564388	2551823	2605193	2923248	2624057	2959855	2563352

11 lentelė. Vidutinis patobulinto Šelo algoritmo atliekamų priskyrimų skaičius naudojant vidutinio ilgio tarpų sekas

N	Tokuda	Ciura	S/Y	I/S	Sedgewick	R/B/H/Z	Seka 2
5000	91607	91536	96002	101234	94353	110577	94638
10000	201994	202938	213152	226716	208567	243602	209125
20000	443461	447218	469964	503670	455367	529908	459749
40000	967300	979059	1019801	1120974	991883	1161681	1005877
80000	2099578	2125122	2209278	2479564	2137726	2521194	2179979
100000	2691737	2709969	2831031	3181857	2741611	3209010	2796455

12 lentelė. Vidutinis patobulinto Šelo algoritmo veikimo laikas naudojant vidutinio ilgio tarpų sekas

N	Tokuda	Ciura	S/Y	I/S	Sedgewick	R/B/H/Z	Seka 2
5000	289	284	307	325	335	300	301
10000	758	773	732	753	666	601	716
20000	1559	1635	1594	1628	1647	1468	1523
40000	3810	3483	3426	3630	3451	3124	3441
80000	8956	8875	8077	9163	9153	7673	8767
100000	11080	11200	11228	12138	12006	10361	11283

Išvados

Gavome tą ir aną, rekomenduojame šitai.

Literatūra

- [Aut09] The Go Authors. Sort implementation in Go standard library. 2009. URL: <https://golang.org/src/sort/sort.go> (tikrinta 2021-05-24).
- [BFM06] Michael A Bender, Martin Farach-Colton ir Miguel A Mosteiro. Insertion sort is $O(n \log n)$. *Theory of Computing Systems*, 39(3):391–397, 2006.
- [Ciu01] Marcin Ciura. Best increments for the average case of shellsort. *International Symposium on Fundamentals of Computation Theory*, p.p. 106–117. Springer, 2001.
- [CK80] Curtis R. Cook ir Do Jin Kim. Best Sorting Algorithm for Nearly Sorted Lists. *Commun. ACM*, 23(11):620–624, 1980-11. ISSN: 0001-0782. DOI: 10.1145/359024.359026. URL: <https://doi.org/10.1145/359024.359026>.
- [FG15] Neetu Faujdar ir Satya Prakash Ghrera. Analysis and Testing of Sorting Algorithms on a Standard Dataset. *2015 Fifth International Conference on Communication Systems and Network Technologies*, p.p. 962–967, 2015. DOI: 10.1109/CSNT.2015.98.
- [For64] G. E. Forsythe. Algorithms. *Commun. ACM*, 7(6):347–349, 1964-06. ISSN: 0001-0782. DOI: 10.1145/512274.512284. URL: <https://doi.org/10.1145/512274.512284>.
- [GK72] David Gale ir Richard M. Karp. A phenomenon in the theory of sorting. *Journal of Computer and System Sciences*, 6(2):103–115, 1972. ISSN: 0022-0000. DOI: [https://doi.org/10.1016/S0022-0000\(72\)80016-3](https://doi.org/10.1016/S0022-0000(72)80016-3). URL: <https://www.sciencedirect.com/science/article/pii/S0022000072800163>.
- [HGL⁺06] Gregory Hornby, Al Globus, Derek Linden ir Jason Lohn. Automated antenna design with evolutionary algorithms. *Space 2006*, p. 7242. 2006.
- [Hol92] John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.
- [IS85] Janet Incerpi ir Robert Sedgewick. Improved upper bounds on Shellsort. *Journal of Computer and System Sciences*, 31(2):210–224, 1985.
- [Knu70] Donald E. Knuth. Von Neumann’s First Computer Program. *ACM Comput. Surv.*, 2(4):247–260, 1970-12. ISSN: 0360-0300. DOI: 10.1145/356580.356581. URL: <https://doi.org/10.1145/356580.356581>.
- [Knu72] Donald E Knuth. Ancient babylonian algorithms. *Communications of the ACM*, 15(7):671–677, 1972.
- [MAM⁺17] Arash Mohammadi, Houshyar Asadi, Shady Mohamed, Kyle Nelson ir Saeid Nahavandi. OpenGA, a C++ Genetic Algorithm Library. *Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on*, p.p. 2051–2056. IEEE, 2017.

- [PPS92] C. G. Plaxton, B. Poonen ir T. Suel. Improved lower bounds for Shellsort. *Proceedings., 33rd Annual Symposium on Foundations of Computer Science*, p.p. 226–235, 1992. DOI: 10.1109/SFCS.1992.267769.
- [Pra72] Vaughan R Pratt. Shellsort and sorting networks. Tech. atask., STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1972.
- [RB13] Irmantas Radavičius ir Mykolas Baranauskas. An empirical study of the gap sequences for Shell sort. *Lietuvos matematikos rinkinys*, 54(A):61–66, 2013-12. DOI: 10.15388/LMR.A.2013.14. URL: <https://www.journals.vu.lt/LMR/article/view/14899>.
- [RBH⁺02] Robert S Roos, Tiffany Bennett, Jennifer Hannon ir Elizabeth Zehner. A genetic algorithm for improved shellsort sequences. *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, p.p. 694–694, 2002.
- [Sed86] Robert Sedgewick. A new upper bound for Shellsort. *Journal of Algorithms*, 7(2):159–173, 1986. ISSN: 0196-6774. DOI: [https://doi.org/10.1016/0196-6774\(86\)90001-5](https://doi.org/10.1016/0196-6774(86)90001-5). URL: <https://www.sciencedirect.com/science/article/pii/0196677486900015>.
- [Sew10] Julian Seward. Sort implementation in bzip2. 2010. URL: https://www.ncbi.nlm.nih.gov/IEB/ToolBox/CPP_DOC/lxr/source/src/util/compress/bzip2/blocksort.c#L519 (tikrinta 2021-05-24).
- [She59] D. L. Shell. A High-Speed Sorting Procedure. *Commun. ACM*, 2(7):30–32, 1959-07. ISSN: 0001-0782. DOI: 10.1145/368370.368387. URL: <https://doi.org/10.1145/368370.368387>.
- [SY99] Richard Simpson ir Shashidhar Yachavaram. Faster shellsort sequences: A genetic algorithm application. *Computers and Their Applications*, p.p. 384–387, 1999.
- [Tok92] Naoyuki Tokuda. An Improved Shellsort. *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture - Information Processing '92, Volume I - Volume I*, p.p. 449–457, NLD. North-Holland Publishing Co., 1992. ISBN: 044489747X.
- [Tur37] Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.
- [Wei91] Mark Allen Weiss. Short Note: Empirical study of the expected running time of Shellsort. *The Computer Journal*, 34(1):88–91, 1991.
- [WWG51] Maurice V Wilkes, David J Wheeler ir Stanley Gill. *The Preparation of Programs for an Electronic Digital Computer: With special reference to the EDSAC and the Use of a Library of Subroutines*. Addison-Wesley, 1951.