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

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 mašininiu 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 diskretaus uždavinio pavyzdžių yra duomenų rikiavimas.

Duomenų rikiavimas yra vienas pamatinių informatikos uždavinių. Matematiškai jis formuluojamas taip: duotai baigtinei palyginamų elementų sekai $S = (s_1, s_2, ..., s_n)$ 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 $S_1, S_2, ..., S_n$, kur kiekvienas posekis $S_i = (s_i, s_{i+h}, s_{i+2h}, ...)$ 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^p3^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 euristikas, 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, tikėtinai 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 [SY99] [RBH+02]. 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:

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

Šis darbas sudarytas iš 4 skyrių. Pirmame skyriuje nustatomi kriterijai tarpų sekų efektyvumui įvertinti. Antrame skyriuje paruošiama eksperimentų vykdymo aplinka. Trečiame skyriuje generuojamos tarpų sekos, naudojant genetinius algoritmus. Ketvirtame skyriuje atliekant eksperimentus įvertinamas sugeneruotų ir pateiktų literatūroje tarpų sekų efektyvumas.

1. Tarpų sekų efektyvumo kriterijų nustatymas

Šelo algoritmo tarpų sekų efektyvumo įvertinimas nėra trivialus. Rikiavimo algoritmai dažniausiai yra vertinami pagal atliekamų priskyrimų skaičių. Skaičiuojant algoritmo atliekamus priskyrimus, gana paprasta jais išreikšti inversijų skaičių. Daugelyje algoritmų priskyrimų skaičius uždaviniui augant greitai artėja prie palyginimų skaičiaus, tad tokiu metodu gautas įvertis būna pakankamai tikslus. Šelo algoritmo atveju, vien priskyrimų skaičius nėra pakankamai tikslus kriterijus tarpų sekų efektyvumui įvertinti, kadangi remiantis tik juo, gautas įvertis neatspindi praktinio efektyvumo. Kaip parodo [Ciu01], šiame algoritme dominuojanti operacija yra palyginimas. Tad galima daryti išvadą, jog atliekamų palyginimų skaičius yra tinkamesnis kriterijus efektyvumui įvertinti.

Matuojant rikiavimo algoritmo efektyvumą tik naudojant sveikaskaitinius pradinius duomenis, gauti rezultatai gali būti netikslūs, kadangi šių 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 kurie saugomi steke, priskyrimas reikalauja perkopijuoti visą įrašą, o palyginimas gali būti atliekamas naudojant tam tikrą raktą. 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 praktinio 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.

2. Eksperimentų vykdymo aplinkos paruošimas

2.1. Techninės detalės

Tyrimui 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. Genetinio algoritmo implementacijai buvo pasirinkta OpenGA biblioteka [MAM+17] dėl suteikiamos laisvės pasirinkti, kaip įgyvendinti genetinius operatorius bei modernių kalbos konstruktų ir lygiagretaus vykdymo palaikymo.

2.2. Šelo rikiavimo algoritmo variantai

Kaip ir daugelis algoritmų, Šelo rikiavimo algoritmas turi keletą variantų. Kaip pabrėžia [RB13], vadovėlinė šio algoritmo implementacija nėra naši, kadangi atlieka nereikalingus prisky-rimus, todėl buvo pasirinkta atlikti matavimus naudojant ir vadovėlinę, ir patobulintą šio algoritmo versiją. Abiejų algoritmų pseudokodas pateikiamas žemiau.

Algorithm 1 Vadovėlinis Šelo rikiavimo algoritmas

```
1:  for each gap in H do
2:      for i ← gap + 1 to N do
3:          j ← i
4:          temp ← S[i]
5:          while j > gap and S[j − gap] > S[j] do
6:              S[j] ← S[j − gap]
7:              j ← j − gap
8:          end while
9:          S[j] ← temp
10:     end for
11: end for
```

Algorithm 2 Patobulintas Šelo rikiavimo algoritmas

```
1:  for each gap in H do
2:      for i ← gap + 1 to N do
3:          if S[i − gap] > S[i] then
4:              j ← i
5:              temp ← S[i]
6:              repeat
7:                  S[j] ← S[j − gap]
8:                  j ← j − gap
9:              until j ≤ gap or S[j − gap] ≤ S[j]
10:             S[j] ← temp
11:         end if
12:     end for
13: end for
```

2.3. Pasiruošimas matavimų atlikimui

Siekant išmatuoti abiejų algoritmų atliekamų operacijų skaičių, buvo parengtos modifikuotos jų versijos, kurios kaip rezultatą grąžina atliktus palyginimus bei priskyrimus. Tiesa, matuoti algoritmų veikimo laiką naudojant šias versijas nėra tinkama, kadangi jos atlieka papildomus žingsnius ir gali iškreipti gautus rezultatus. Atsižvelgiant į tai, veikimo laiko matavimui naudotos nemodifikuotos abiejų algoritmų implementacijos.

Algorithm 3 Vadovėlinis operacijas skaičiuojantis Šelo rikiavimo algoritmas

```
1:  cmp ← 0
2:  asn ← 0
3:  for each gap in H do
4:      for i ← gap + 1 to N do
5:          cmp ← cmp + 1
6:          j ← i
7:          temp ← S[i]
8:          asn ← asn + 2
9:          while j > gap and S[j − gap] > S[j] do
10:             S[j] ← S[j − gap]
11:             j ← j − gap
12:             asn ← asn + 2
13:             cmp ← cmp + 2
14:         end while
15:         if j < gap then
16:             cmp ← cmp + 1
17:         else
18:             cmp ← cmp + 2
19:         end if
20:         S[j] ← temp
21:         asn ← asn + 1
22:     end for
23: end for
24: return (asn, cmp)
```

Algorithm 4 Patobulintas operacijas skaičiuojantis Šelo rikiavimo algoritmas

```
1: for each gap in H do
2:     for i ← gap + 1 to N do
3:         if S[i − gap] > S[i] then
4:             j ← i
5:             temp ← S[i]
6:             repeat
7:                 S[j] ← S[j − gap]
8:                 j ← j − gap
9:             until j ≤ gap or S[j − gap] ≤ S[j]
10:            S[j] ← temp
11:        end if
12:    end for
13: end for
```

3. Sekų generavimas

3.1. Genetinis algoritmas

3.2. Galinių sekų generavimas

Pirmame sekų generavimo etape buvo generuojamos tarpų sekos, kurios efektyvios kai $N = 1000$. Remiantis [SY99], tokios sekos chromosoma buvo modeliuojama kaip septynių sveikų skaičių masyvas. Toliau šios sekos vadinamos galinėmis sekomis. Galinių sekų generavimui buvo paruoštas vienkriterinis genetinis algoritmas. 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 vaikinei chromosomai. Mutacijos operatorius buvo įgyvendintas su $\frac{1}{2}$ tikimybe keičiant kiekvieną chromosomos geną, pridedant prie jo atsitiktinį skaičių iš intervalo $[-100, 100]$ su apribojimu, jog pakeistas genas turi priklausyti intervalui $[1, 1000]$.

Pirmiausia algoritmas buvo vykdomas su atsitiktinai sugeneruota 10000 individų populiacija, kiekvieną iteraciją pritaikant mutacijos ir rekombinacijos operatorius atitinkamai $\frac{8}{10}$ ir $\frac{1}{10}$ populiacijos. Tokiu būdu buvo sugeneruota ir atrinkta 10 sekų. Po to algoritmas buvo vykdomas su atsitiktinai sugeneruota 15000 individų populiacija, dalį jos inicializuojant geriausiomis prieš tai gautomis sekomis ir kiekvieną iteraciją pritaikant mutacijos ir rekombinacijos operatorius atitinkamai $\frac{9}{10}$ ir $\frac{1}{100}$ populiacijos. Tokiu būdu buvo sugeneruota ir atrinkta dar 10 sekų. Galiausiai iš visų sugeneruotų sekų buvo atrinkta tinkamiausia. Tai buvo atlikta sugeneruotas sekas naudojant 100000 atsitiktinai sugeneruotų masyvų rikiavimui vadovėline Šelo rikiavimo algoritmo versija ir matuojant atliktų palyginimo operacijų aritmetinį vidurkį. Atliekant matavimus taip pat buvo įtraukta Simpson ir Yachavaram pasiūlyta [SY99] seka 1, 4, 13, 36, 83, 219, 893. Tai buvo atlikta todėl, kad autorių teigimu tai sparčiausiai veikianti jų rasta galinė seka ir ji suteikė puikų kontekstą įvertinti darbo metu sugeneruotų galinių sekų tinkamumą. Geriausiai pasirodė seka 1, 5, 13, 40, 137, 669, 974.

4. Sekų efektyvumo įvertinimas

placeholder....

Išvados

Gavome tą ir aną, rekomenduojame šitai.

Literatūra

[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.

[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. 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.

[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.

[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 1 - Volume I*, p.p. 449–457, NLD. North-Holland Publishing Co., 1992. ISBN: 044489747X.

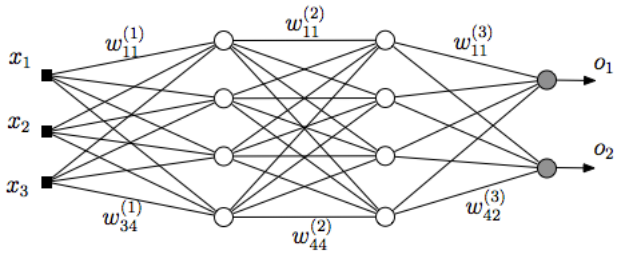
[Tur37] Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

[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.

Priedas Nr. 1

Paveikslėlis



1 pav. Paveikslėlio pavyzdys