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. Apibendrinant, sunku įsivaizduoti informatikos mokslą be algoritmo sąvokos.

Vienas žinomiausių diskretaus uždavinio pavyzdžių yra duomenų rikiavimas. 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. Taigi rikiavimas yra vienas pamatinių informatikos uždavinių.

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) [She59a] 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 blogiausiu atveju [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. Todėl ir dabar yra tikslinga ieškoti naujų ir efektyvesnių Šelo algoritmo tarpų sekų.

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. Nesunku pastebėti, jog genetiniai algoritmai gali palengvinti tam tikrų rūšių uždavinių sprendimą.

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 [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.
- Atlikti genetinių algoritmų 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 atliekama genetinių algoritmų literatūros analizė. Trečiame 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

Šis skyrius sudarytas iš 2 poskyrių. Pirmame poskyryje nagrinėjama Šelo algoritmo veikimo sparta. Antrame poskyryje analizuojamos Šelo algoritmo versijos.

## 1.1. Šelo rikiavimo algoritmo veikimo sparta

Šiame poskyryje nagrinėjamas Šelo algoritmo veikimo greitis priklausomai nuo duomenų dydžio ir duomenų specifikos.

### 1.1.1. Veikimo greitis priklausomai nuo duomenų dydžio

Renkantis kokį algoritmą naudoti, labai svarbu įvertinti tikėtiną duomenų dydį. Šelo rikiavimo algoritmas veikia greičiausiai, kai duomenų dydis yra ganėtinai mažas [Ciu01]. Kaip teigiama [SY99], ši savybė galioja, kai $N \leq 106$. Tokiu atveju, Šelo algoritmas 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 labai greitas, kai uždavinys didelis, yra scenarijų, kuriais jį rinktis tikrai verta.

### 1.1.2. Veikimo greitis 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. Moksliniuose tyrimuose ši savybė retai turi įtakos, kadangi dažniausia siekiama ištirti algoritmo veikimą, kai duomenys nėra tvarkingai išdėstyti ir tipiškai renkamasi atsitiktinai generuoti pradinius duomenis šiam tikslui pasiekti. Savaime 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 veikia žymiai greičiau.

## 1.2. Šelo rikiavimo algoritmo versijos

Šiame poskyryje analizuojami du Šelo algoritmo variantai: vadovėlinis Šelo algoritmas ir patobulintas Šelo algoritmas.

### 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), kurią D. L. Shell paskelbė dar 1959 metais [She59b]. Jos pseudokodas pateikiamas 1 algoritme.

Įdėmiau įsižiūrėjus, nesunku pastebėti Šelo algoritmo panašumą į paprastą rikiavimą įterpimu - pašalinus išorinį ciklą, iteruojantį tarpų sekos narius bei pakeitus kintamąjį $gap$ į $1$, gaunamas būtent šio algoritmo kodas. Paties rikiavimo įterpimu idėja yra gana paprasta: pradžioje rikiuojama seka yra suskaidoma į du posekius, kur kairysis (susidedantis iš kairiausio sekos elemento) yra išrikiuotas, o dešinysis (susidedantis iš visų likusių sekos elementų) - ne. Tada kiekvieną iteraciją išrikiuotas posekis yra plečiamas, imant kairiausią neišrikiuoto posekio elementą ir įterpiant jį į išrikiuotą posekį. Naujų elementų įterpimas į išrikiuotą posekį vyksta iteratyviai sukeičiant įterpiamą elementą su elementu, esančiu jam iš kairės, jei šis yra didesnis už įterpiamą elementą. Algoritmas baigia darbą, kai neišrikiuoto posekio ilgis yra 0.

Verta pastebėti, jog Šelo algoritmas tėra rikiavimo įterpimu generalizacija, kur įterpimas gali vykti atstumu, didesniu nei 1. Būtent didesnis įterpimo atstumas ir suteikia Šelo algoritmui pranašumą, kadangi labiausiai nuo galutinės pozicijos nutolę elementai gali greičiau patekti į reikiamą poziciją. Šelo algoritmo idėjos turi tvirtą matematinį pagrindą: rikiavimas įterpimu turi $O(n^2)$ laiko sudėtingumą blogiausiu atveju (pavyzdžiui, kai rikiuojama seka yra išrikiuota atgaline tvarka), tačiau kai didžiausias atstumas tarp dviejų neišrikiuotų elementų yra $k$, rikiavimo įterpimu laiko sudėtingumas yra $O(kn)$. Taigi seką $h$-išrikiavus, paskutinė iteracija turės $O(hn)$ laiko sudėtingumą. Iteracijos su dideliais tarpais taip pat yra gana greitos, kadangi dirbama su daug mažesniais elementų kiekias, o šiuo atveju rikiavimas įterpimu yra vienas greičiausių.

**Algorithm 1** Vadovėlinis Šelo rikiavimo algoritmas

```
1: foreach 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
```

### 1.2.2. Patobulintas Šelo rikiavimo algoritmas

Verta pastebėti, jog vadovėlinė Šelo algoritmo implementacija nėra pilnai efektyvi [RB13]. Vykdant vidinį vadovėlinio Šelo algoritmo ciklą (1 algoritmo 5-8 eilutės), 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 4 ir 9 eilutėse vis tiek yra vykdomi du priskyrimai, kurie šiuo atveju nueina veltui (o taip nutinka pakankamai dažnai). Siekiant tai ištaisyti, [RB13] pateikė patobulintą Šelo algoritmo versiją (toliau - PŠA), kurią galima rasti 2 algoritme.

PŠA prieš vykdydamas bet kokius kitus žingsnius patikrina, ar elementas $S[j]$ jau yra tinkamoje pozicijoje (žr. 2 algoritmo 3 eil.), kas leidžia sumažinti atliekamų priskyrimų skaičių, kai su $S[j]$ pagrindinės rikiavimo logikos (2 algoritmo 4-10 eil.) nėra reikalo vykdyti. Taip pat, kadangi iš pat pradžių yra užtikrinama, jog galima atlikti nors vieną sukeitimą, vadovėlinio Šelo algoritmo $while$ ciklą (1 algoritmo 5-8 eil.) galima pakeisti $repeat \ldots until$ tipo ciklu, kas ir yra atliekama 2 algoritmo 6-9 eilutėse.

Reikia pastebėti, jog patobulintas Šelo algoritmas nuo vadovėlinės versijos atliekamų operacijų prasme skiriasi tik atliekamais priskyrimais. Tačiau šios optimizacijos svarba pasirodo pakankamai didelė - autorių teigimu [RB13], PŠA vidutiniškai atlieka 40-80% mažiau priskyrimų ir veikia 20% greičiau, nei VŠA. Tiesa, PŠA nėra plačiai naudojamas, o jo veikimas vidutiniu atveju originaliame darbe ištirtas tik rėžyje $500 \leq N \leq 2000$.

**Algorithm 2** Patobulintas Šelo rikiavimo algoritmas

```
1: foreach 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. Genetiniai algoritmai

Paprasčiausias genetinis algoritmas susideda iš chromosomų populiacijos bei atrankos, mutacijos ir rekombinacijos operatorių [SY99]. Šiame skyriuje bus nagrinėjamos šių terminų reikšmės ir genetinių algoritmų veikimo principai.

## 2.1. Chromosomų populiacija

Chromosoma GA kontekste vadiname potencialų uždavinio sprendinį. Projektuojant genetinį algoritmą tam tikro uždavinio sprendimui, svarbu tinkamai pasirinkti, kaip kompiuteriu modeliuoti galimus sprendinius (chromosomas). Chromosomos kompiuterio atmintyje standartiškai išreiškiamos bitų eilutėmis [Whi94], 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.

Sprendinio kokybę įvardijame kaip jo tinkamumą, kuris apibrėžiamas tinkamumo funkcijos reikšme, pateikus sprendinį kaip parametrą. 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.

Chromosomų rinkinys, literatūroje dažnai vadinamas populiacija, atspindi uždavinio sprendinių aibę, kuri kinta kiekvieną genetinio algoritmo iteraciją. Populiaciją dažnu atveju sudaro šimtai ar net tūkstančiai individų. Tiesa, populiacijos dydį būtina nustatyti priklausomai nuo sprendžiamo uždavinio, kadangi per didelės populiacijos pasirinkimas chromosomų kokybės gali nepadidinti ir privesti prie resursų švaistymo. Jei GA suprojektuotas tinkamai, dažnu atveju vidutinis populiacijos tinkamumas gerės kiekvieną iteraciją [SY99].

## 2.2. Genetiniai operatoriai

Esminė GA dalis yra populiacijos genetinės įvairovės užtikrinimas, geriausių individų atranka ir kryžminimasis. Šiems procesams nesunku rasti atitikmenis nagrinėjant gamtoje vykstančius evoliucinius reiškinius. Siekiant užtikrinti šių biologinių procesų išpildymą, genetinis algoritmas vykdymo metu iteratyviai atnaujina esamą populiaciją ir kuria naujas kartas taikydamas biologijos žiniomis paremtus atrankos, rekombinacijos ir mutacijos operatorius.

Atrankos operatorius grąžina tinkamiausius populiacijos individus, kuriems yra leidžiama susilaukti palikuonių taikant rekombinacijos operatorių. Dažniausiai atranka vykdoma atsižvelgiant į populiacijos individų tinkamumą, atrenkant ir pateikiant rekombinacijai tuos, kurių tinkamumas yra geriausias. Verta pastebėti, jog įprastai rekombinacijai yra pasirenkama tam tikra fiksuota einamosios populiacijos dalis ir daugelyje GA implementacijų šis dydis yra nurodomas kaip veikimo parametras. Įprastai populiacijos dalis, kuri nebuvo atrinkta rekombinacijai, būna mažesnė. Daž-

niausiai ir jai priklausančios chromosomos yra perkeliamos į kitą GA iteraciją, siekiant užtikrinti naujos populiacijos genetinę įvairovę. Tinkamas rekombinacijai atrenkamos populiacijos dalies pasirinkimas yra svarbus, kadangi kai šis parametras per didelis, populiacija pradeda konverguoti per anksti ir nėra atliekama pakankamai plati galimų sprendinių paieška.

Rekombinacijos operatorius įprastai veikia iš dviejų tėvinių chromosomų sukurdamas naują vaikinę chromosomą, kas dažniausiai pasiekiama tam tikru būdu perkopijuojant tėvų genų atkarpas į vaikinę chromosomą. Tiesa, kai kurie tyrimai [ERR94; Tin05] rodo, jog rekombinacijai naudojant daugiau nei du tėvus, gautų vaikinių chromosomų kokybė yra geresnė. Šiame darbe paprastumo dėlėi bus laikoma, jog rekombinacija vyksta naudojant tik du tėvus. 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. Šiai problemai spręsti yra pasitelkiamos specializuotos rekombinacijos strategijos [LKM+96] arba įvedama chromosomų normalizacija.

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 lokaliojo 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. Įprastai mutacija kiekvienai chromosomai taikoma su tam tikra tikimybe, kuri nurodoma kaip vienas iš GA veikimo parametrų. Tinkamas chromosomos mutacijos tikimybės parinkimas yra vienas iš svarbiausių sprendimų projektuojant GA, kadangi nuo mutacijos tikimybės dažnu atveju priklauso gaunamų sprendinių kokybė. Jei mutacijos tikimybė yra per didelė, GA išsigimsta į primityvią atsitiktinę paiešką [HAA+19] ir rizikuojama prarasti geriausius sprendinius. Jei mutacijos tikimybė per maža, tai gali vesti prie genetinio dreifo [Mas11], kas reiškia, jog populiacijos genetinė įvairovė palaipsniui mažės.

## 2.3. Genetinių algoritmų veikimo principai

Apjungiant visas aukščiau aptartas genetinio algoritmo dalis, galima suformuluoti paprasto genetinio algoritmo pseudokodą. Jis pateikiamas 3 algoritme. Kintamieji $N$ (populiacijos dydis), $mutation\_rate$ (mutacijos tikimybė) bei $recombination\_frac$ (rekombinacijai pateikiamų chromosomų dalis) tipiškai būtų nustatomi vartotojo ir perduodami kaip parametrai, tačiau paprastumo dėlei pakanka laikyti, jog jie nustatomi paties GA. Algoritmui pradedant darbą yra inicializuojama pradinė chromosomų populiacija (žr. 1 eil.). Vykdant pagrindinį GA ciklą, pirmiausia yra įvertinamas einamosios populiacijos tinkamumas (žr. 3 eil.). Tada iš tinkamiausių populiacijos individų yra atsitiktinai atrenkamos dvi tėvinės chromosomos, kurioms taikant rekombinacijos operatorių

yra kuriami nauji individai (žr. 7-8 eil.). Naujam individui su tam tikra tikimybe taip pat yra taikomas mutacijos operatorius (žr. 9-11 eil.). Naujų individų generavimas yra kartojamas, kol naujos populiacijos dydis siekia $N$ (žr. 6-13 eil.). Galiausiai einamajai populiacijai yra priskiriama naujai sugeneruota populiacija (žr. 14 eil.), kas užtikrina, jog kitą ciklo iteraciją bus dirbama su naujais individais. Prieš pradedant naują ciklą (žr. 15 eil.), GA patikrina, ar kažkuri iš sustojimo sąlygų yra patenkinta, ir jei taip - baigia darbą. Praktikoje dažnai siekiama, jog GA sustotų, kai yra pasiektas maksimalus kartų skaičius ar vidutinio tinkamumo pokytis tarp paskutinių dviejų populiacijos kartų yra pakankamai mažas. Tokie sustojimo kriterijai nurodomi, jog GA nenaudotų per daug resursų, ypač jei galima nuspėti, jog tolesnis veikimas neduos geresnių rezultatų. Žinoma, pateiktas GA yra pakankamai paprastas ir praleidžia daugelį detalių, tačiau autorius mano, jog jis tinkamai iliustruoja pagrindinius genetinių algoritmų veikimo principus.

**Algorithm 3** Paprastas GA

1: Let $current\_population$ be a random population of chromosomes.
2: **repeat**
3: Apply the fitness function to each chromosome in $current\_population$.
4: Sort the chromosomes in $current\_population$ based on their fitness in descending order.
5: Let $new\_population$ be an empty set.
6: **repeat**
7: Let $(parent1, parent2)$ be a pair of parent chromosomes selected randomly from the first $[N * recombination\_frac]$ elements of $current\_population$.
8: Let $child\_chromosome$ be the result of applying the recombination operator to $parent1$ and $parent2$.
9: **if** $rand01() \leq mutation\_rate$ **then**
10: Apply the mutation operator to $child\_chromosome$
11: **end if**
12: Add $child\_chromosome$ to $new\_population$
13: **until** N offspring have been created.
14: Assign $new\_population$ to $current\_population$.
15: **until** Some termination criterion is satisfied.

# 3. 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.

Autoriaus nuomone, atliktų palyginimų skaičius yra svarbiausias kriterijus tarpų sekų efektyvumui įvertinti ir didžiausias svoris atliekant bendrą visų kriterijų vertinimą turėtų būti teikiamas būtent jam. Vertinant kitus du kriterijus, didesnis svoris turėtų atitekti atliekamų palyginimų skaičiui, kadangi veikimo laiko įvertis nėra universalus. Verta pastebėti, jog atliktų priskyrimų skaičių reikia vertinti, atsižvelgiant į kontekstą, kadangi seka, kuri atlieka daug priskyrimų, nebūtinai yra neefektyvi - daug svarbesnis yra atliekamų palyginimų ir priskyrimų santykis. Tai galima paaiškinti paprastu pavyzdžiu: jei su tam tikra tarpų seka Šelo algoritmas atlieka santykinai mažai palyginimų ir daug priskyrimų, galima laikyti, jog su ja algoritmas dirba pakankamai efektyviai, kadangi atlikti palyginimai nenueina veltui ir rikiuojamos sekos $S$ elementai greitai artėja link galutinės pozicijos. Reikia atsižvelgti ir į tai, jog vadovėlinė Šelo algoritmo implementacija dažnai atlieka nereikalingus priskyrimus [RB13], tad atliekant sekų vertinimą, yra tikslinga įvertinti sekas naudojant ir VŠA, ir PŠA. Savaime suprantama, jog į veikimo laiką verta žvelgti kritiškai, kadangi jis priklauso nuo algoritmo implementacijos, eksperimentams naudojamos mašinos techninių parametrų, operacinės

sistemos, pasirinktos programavimo kalbos ar net kompiliatoriaus versijos. Todėl vertinant veikimo laiką, svarbiausia atsižvelgti į santykinius gautų rezultatų skirtumus naudojant skirtingas tarpų sekas.

# 4. Tarpų sekų generavimas

Šį skyrių sudaro 2 poskyriai. Pirmame poskyryje naudojant genetinius algoritmus generuojamos trumpos tarpų sekos. Antrame poskyryje naudojant genetinius algoritmus generuojamos vidutinio ilgio tarpų sekos.

## 4.1. 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 atsitiktinai sugeneruotų sveikų skaičių masyvų, turinčių po 1000 elementų, 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. GA sustojimo kriterijumi buvo pasirinkta laikyti tinkamiausio individo nepakitimą 25 iteracijas. Mutacijos operatorius buvo įgyvendintas su $\frac{1}{5}$ tikimybe keičiant tam tikrą chromosomą, pridedant prie atsitiktinai pasirinkto geno reikšmės skaičių $\left[(rand01() - rand01()) * \frac{25}{\sqrt{karta}}\right]$ 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, turinčių po 1000 elementų, rikiavimui ir matuojant atliktų palyginimo operacijų aritmetinį vidurkį. Atlikus matavimus, buvo pasirinkta seka $855, 264, 86, 35, 12, 5, 1$ (toliau - S1).

## 4.2. 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 12 sveikų skaičių masyvas. 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 100000 elementų masyvai. GA buvo nurodyta sustoti, kai kartų skaičius pasieks 100. Mutacijos operatorius buvo pakoreguotas ir su $\frac{1}{5}$ tikimybe keitė tam tikrą chromosomą,

pridedant prie atsitiktinai pasirinkto geno reikšmės skaičių $\left[(rand01() - rand01()) * \frac{75}{\sqrt{karta}}\right]$.

GA buvo vykdomas su atsitiktinai sugeneruota 1000 individų populiacija, paskutinius septynis kiekvienos 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 - didžiausią tinkamumą turinčios 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 rikiuojamų masyvų dydį į 100000. Atlikus matavimus, buvo pasirinkta seka $45794, 17396, 7414, 3136, 1206, 561, 264, 86, 35, 12, 5, 1$ (toliau - S2).

# 5. Eksperimentų vykdymo aplinkos paruošimas

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

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.

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

Eksperimentų vykdymui buvo naudojamas 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.

# 6. Tarpų sekų efektyvumo įvertinimas

Šį skyrių sudaro 2 poskyriai. Pirmame poskyryje atliekant eksperimentus yra įvertinamas trumpų tarpų sekų efektyvumas. Antrame poskyryje atliekant eksperimentus yra įvertinamas vidutinio ilgio tarpų sekų efektyvumas.

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 ilgius, 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.

Remiantis prielaida, jog tiksliam tarpų sekų efektyvumo įvertinimui reikalinga jas ištirti naudojant ir VŠA, ir PŠA, tai ir buvo atlikta.

## 6.1. Trumpų tarpų sekų efektyvumo įvertinimas

Trumpų tarpų sekų efektyvumo tyrimui buvo pasirinktos šios tarpų sekos:

- Tokuda: $525, 233, 103, 46, 20, 9, 4, 1$ [Tok92]
- Ciura: $701, 301, 132, 57, 23, 10, 4, 1$ [Ciu01]
- Simpson-Yachavaram: $893, 219, 83, 36, 13, 4, 1$ [SY99]
- Sedgewick: $929, 505, 209, 109, 41, 19, 5, 1$ [Sed86]
- Incerpi-Sedgewick: $861, 336, 112, 48, 21, 7, 3, 1$ [IS85]
- S1: $855, 264, 86, 35, 12, 5, 1$

Atsižvelgiant į santykinai nedidelį duomenų dydį, matavimai buvo atlikti 50000 kartų. Gauti duomenys pateikiami suapvalinus iki šimtosios dalies, 3 geriausi rezultatai su tam tikru $N$ pateikiami pajuodintu šriftu.

### 6.1.1. Efektyvumo tyrimas naudojant vadovėlinį Šelo algoritmą

1 lentelė. Vidutinis VŠA atliktų palyginimų skaičius naudojant trumpas tarpų sekas

| N | Tokuda | Ciura | S/Y | Sedgewick | I/S | S1 |
|---|---|---|---|---|---|---|
| 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 VŠA atliktų priskyrimų skaičius naudojant trumpas tarpų sekas

| N | Tokuda | Ciura | S/Y | Sedgewick | I/S | S1 |
|---|---|---|---|---|---|---|
| 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 VŠA veikimo laikas (µs) naudojant trumpas tarpų sekas

| N | Tokuda | Ciura | S/Y | Sedgewick | I/S | S1 |
|---|---|---|---|---|---|---|
| 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** |

Kaip galima pastebėti iš 1 lentelės, mažiausiai palyginimų bendru atveju atliko Ciura seka, nedaug atsiliko S1 ir Simpson-Yachavaram sekos. Mažiausiai priskyrimų, remiantis 2 lentele, atliko S1 seka, nežymiai daugiau - Simpson-Yachavaram ir Sedgewick sekos. Vertinant vidutinį veikimo laiką pateiktą 3 lentelėje, sunku išskirti geriausią, tačiau palankiausi rezultatai gauti su Sedgewick ir Simpson-Yachavaram sekomis bei S1 seka. Todėl, įvertinant visus kriterijus, šiame etape geriausiai pasirodė Ciura, S1 ir Simpson-Yachavaram sekos.

### 6.1.2. Efektyvumo tyrimas naudojant patobulintą Šelo algoritmą

4 lentelė. Vidutinis PŠA atliktų palyginimų skaičius naudojant trumpas tarpų sekas

| N | Tokuda | Ciura | S/Y | Sedgewick | I/S | S1 |
|---|---|---|---|---|---|---|
| 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 PŠA atliktų priskyrimų skaičius naudojant trumpas tarpų sekas

| N | Tokuda | Ciura | S/Y | Sedgewick | I/S | S1 |
|---|---|---|---|---|---|---|
| 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 PŠA veikimo laikas (µs) naudojant trumpas tarpų sekas

| N | Tokuda | Ciura | S/Y | Sedgewick | I/S | S1 |
|---|---|---|---|---|---|---|
| 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** |

Analizuojant atliktų palyginimų skaičių, pateikiamą 4 lentelėje, rezultatai beveik nesiskiria nuo VŠA ir geriausiai pasirodė Ciura, S1 ir Simpson-Yachavaram sekos. Vertinant atliktų priskyrimų skaičių, rezultatai labai stipriai skiriasi nuo gautų naudojant VŠA. Kaip matoma 5 lentelėje, mažiausiai priskyrimų atlieka Tokuda ir Ciura sekos. Geriausiai šiame etape vidutinio veikimo laiko atžvilgiu, remiantis 6 lentele, pasirodė Sedgewick ir Simpson-Yachavaram sekos bei S1 seka. Įvertinant visus kriterijus, geriausi rezultatai šioje tyrimo dalyje gauti su Tokuda, Ciura ir S1 sekomis.

### 6.1.3. Gautų rezultatų apibendrinimas

Verta pastebėti, jog patobulintas Šelo algoritmas atliekamų palyginimų skaičiumi teoriškai neturėtų skirtis nuo vadovėlinės implementacijos, tad gauti įverčiai leidžia įvertinti matavimų tikslumą atspindint Šelo algoritmo atliekamų palyginimų skaičių naudojant šias tarpų sekas vidutiniu atveju. Gauti rezultatai rodo, jog Ciura seka atlieka mažiausiai priskyrimų vidutiniu atveju, kai $N \leq 1000$, kaip ir teigta [Ciu01]. Atsižvelgiant į pastebimai prastesnius veikimo laikus, sunku rekomenduoti PŠA nesudėtingų tipų duomenų rikiavimui, kai $N \leq 1000$. Tiesa, rikiuojant duomenis, kurių priskyrimas brangus, PŠA veikimo laikas tikėtinai būtų geresnis, nei VŠA, tačiau tuo įsitikinti reikėtų atskiro tyrimo. GA naudojimas Šelo algoritmo trumpų tarpų sekų generavimui duoda gerus rezultatus, kadangi abi GA sugeneruotos sekos (Simpson-Yachavaram ir S1) vidutiniu atveju atlieka daugiau palyginimų tik už Ciura seką, yra vienos iš lyderių vertinant atliktų priskyrimų skaičių naudojant VŠA ir pateikia vienus geriausių veikimo laikų. Tiesa, naudojant GA sugeneruotos sekos nėra be trūkumų: lyginant su Tokuda ir Ciura sekomis, kurios atlieka daugiau priskyrimų naudojant VŠA, šios sekos atlieka žymiai daugiau priskyrimų naudojant PŠA, nei Tokuda ir Ciura sekos. Tai sukelia klausimų dėl jų efektyvumo, kadangi PŠA pašalina tik nereikalingai atliekamus priskyrimus. Stebina faktas, jog naudojant VŠA gauti veikimo laikai yra 10-20% geresni kai $N \geq 200$, nei gauti naudojant PŠA, nors su PŠA atliktų operacijų skaičiai yra žymiai mažesni. Taip pat verta pastebėti, jog su PŠA atliktų palyginimų ir priskyrimų skaičiaus santykis artėja prie 1 augant N, priešingai nei naudojant VŠA.

## 6.2. 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 [Tok92]
- Ciura: 90927,40412,17961,7983,3548,1577,701,301,132,57,23,10,4,1 [Ciu01]
- Simpson-Yachavaram: 38291,22927,8992,3568,1488,893,219,83,36,13,4,1 [SY99]
- Incerpi-Sedgewick: 86961,33936,13776,4592,1968,861,336,112,48,21,7,3,1 [IS85]
- Sedgewick: 64769,36289,16001,8929,3905,2161,929,505,209,109,41,19,5,1 [Sed86]
- Roos et al.: 91433,72985,13229,5267,2585,877,155,149,131,23,8,1 [RBH+02]
- S2: 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, 3 geriausi rezultatai su tam tikru $N$ pateikiami pajuodintu šriftu.

### 6.2.1. Efektyvumo tyrimas naudojant vadovėlinį Šelo algoritmą

7 lentelė. Vidutinis VŠA atliktų palyginimų skaičius naudojant vidutinio ilgio tarpų sekas

| N | Tokuda | Ciura | S/Y | I/S | Sedgewick | R/B/H/Z | S2 |
|---|---|---|---|---|---|---|---|
| 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 VŠA atliktų priskyrimų skaičius naudojant vidutinio ilgio tarpų sekas

| N | Tokuda | Ciura | S/Y | I/S | Sedgewick | R/B/H/Z | S2 |
|---|---|---|---|---|---|---|---|
| 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 VŠA veikimo laikas (μs) naudojant vidutinio ilgio tarpų sekas

| N | Tokuda | Ciura | S/Y | I/S | Sedgewick | R/B/H/Z | S2 |
|---|---|---|---|---|---|---|---|
| 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** |

Kaip galima pastebėti 7 lentelėje, mažiausiai palyginimų bendru atveju atliko Ciura ir Tokuda sekos, šiek tiek daugiau - seka S2. Mažiausiai priskyrimų, remiantis 8 lentele, atliko Simpson-Yachavaram ir S2 sekos. Mažiausi veikimo laikai, pateikiami 9 lentelėje, pastebimi su Roos et al. seka, sekos S2 rezultatai taip pat palankūs. Vertinant visus kriterijus, šiame etape geriausiai pasirodė Ciura ir Tokuda sekos bei seka S2.

### 6.2.2. Efektyvumo tyrimas naudojant patobulintą Šelo algoritmą

10 lentelė. Vidutinis PŠA atliktų palyginimų skaičius naudojant vidutinio ilgio tarpų sekas

| N | Tokuda | Ciura | S/Y | I/S | Sedgewick | R/B/H/Z | S2 |
|---|---|---|---|---|---|---|---|
| 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 PŠA atliktų priskyrimų skaičius naudojant vidutinio ilgio tarpų sekas

| N | Tokuda | Ciura | S/Y | I/S | Sedgewick | R/B/H/Z | S2 |
|---|---|---|---|---|---|---|---|
| 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 PŠA veikimo laikas (µs) naudojant vidutinio ilgio tarpų sekas

| N | Tokuda | Ciura | S/Y | I/S | Sedgewick | R/B/H/Z | S2 |
|---|---|---|---|---|---|---|---|
| 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 |

Analizuojant atliktų palyginimų skaičių, pateikiamą 10 lentelėje, rezultatai yra panašūs į gautus tiriant VŠA ir geriausiai pasirodė Ciura, Tokuda ir S2 sekos. Vertinant atliktus priskyrimus, geriausi rezultatai, pateikiami 11 lentelėje, buvo gauti su Tokuda, Ciura ir Sedgewick sekomis. Nagrinėjant vidutinį veikimo laiką, pateikiamą 12 lentelėje, bendru atveju didžiausia sparta gaunama su Roos et al. ir S2 sekomis. Vertinant visus kriterijus, geriausiai pasirodė Ciura ir Tokuda sekos.

### 6.2.3. Gautų rezultatų apibendrinimas

Gauti rezultatai rodo, jog Roos et al. seka, kaip ir teigta [RBH+02], veikia greičiau už Sedgewick ir Simpson-Yachavaram sekas, bent jau kai $N \leq 100000$, o pradiniai duomenys - sveiki skaičiai. Iš tiesų, ši seka pagal gautus rezultatus yra absoliuti lyderė vertinant vidutinį veikimo laiką, tačiau būtina atsižvelgti į faktą, jog ji atlieka labai daug operacijų, tad yra tikėtina, jog rikiuojant duomenis, kurių priskyrimas ar palyginimas yra brangus, gauti veikimo laiko rezultatai nebūtų tokie palankūs. Ciura seka, kaip ir tiriant trumpas tarpų sekas, pagal gautus rezultatus atlieka mažiausiai priskyrimų, kas dalinai pateisina autoriaus spėjimą [Ciu01], jog seką $701,301,132,57,23,10,4,1$ pratęsus bus gauta palyginimų atžvilgiu optimali seka ir didesniems $N$. Simpson-Yachavaram seka, kaip ir teigė [SY99] autoriai, atlieka mažiau priskyrimų ir veikia greičiau, nei Sedgewick seka. GA naudojimas generuojant vidutinio ilgio tarpų sekas duoda gerus rezultatus: nors nė viena iš GA sugeneruotų sekų (Simpson-Yachavaram, Roos et al. ir S2) nelenkia Tokuda ir Ciura pateiktų sekų vertinant atliktus palyginimus, S2 ir Simpson-Yachavaram sekos naudojant VŠA atlieka mažiau priskyrimų, o Roos et al. seka, kaip ir minėta anksčiau, veikia labai greitai. Palyginimų atžvilgiu optimalios sekos (Tokuda ir Ciura) naudojant PŠA veikia pastebimai greičiau, nei naudojant VŠA, ypač augant duomenų dydžiui.

# Išvados

Įvertinant darbe gautus rezultatus, galima teigti, jog genetiniai algoritmai yra tinkamas metodas efektyvių tarpų sekų radimui. Šelo algoritmu rikiuojant nedidelius ($N \leq 1000$) duomenų dydžius, galima rekomenduoti Ciura ir S1 tarpų sekas. Šelo algoritmu rikiuojant didesnius ($1000 \leq N \leq 100000$) duomenų dydžius, galima rekomenduoti Ciura ir Tokuda tarpų sekas bei patobulintą Šelo algoritmo implementaciją.

# 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.

[ERR94] Agoston E Eiben, P-E Raue ir Zs Ruttkay. Genetic algorithms with multi-parent recombination. *International conference on parallel problem solving from nature*, p.p. 78–87. Springer, 1994.

[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. URL: https://www.sciencedirect.com/science/article/pii/S0022000072800163.

[HAA+19] Ahmad Hassanat, Khalid Almohammadi, Esra' Alkafaween, Eman Abunawas, Awni Hammouri ir VB Prasath. Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. *Information*, 10(12):390, 2019.

[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.

[LKM+96] Pedro Larranaga, Cindy MH Kuijpers, Roberto H Murga ir Yosu Yurramendi. Learning Bayesian network structures by searching for the best ordering with genetic algorithms. *IEEE transactions on systems, man, and cybernetics-part A: systems and humans*, 26(4):487–493, 1996.

[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.

[Mas11] Joanna Masel. Genetic drift. *Current Biology*, 21(20):R837–R838, 2011.

[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. 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).

[She59a] 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.

[She59b] Donald L. Shell. A high-speed sorting procedure. *Communications of the ACM*, 2(7):30–32, 1959.

[SY99] Richard Simpson ir Shashidhar Yachavaram. Faster shellsort sequences: A genetic algorithm application. *Computers and Their Applications*, p.p. 384–387, 1999.

[Tin05] Chuan-Kang Ting. On the mean convergence time of multi-parent genetic algorithms without selection. *European Conference on Artificial Life*, p.p. 403–412. Springer, 2005.

[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.

[Whi94] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[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.