

Lygiagretaus programavimo laboratorinių darbų analizė

Pijus Petkevičius

October 22, 2022

Contents

| | |
|--|----------|
| 1. 1 laboratorinis darbas | 3 |
| 1.1. Aprašymas | 3 |
| 1.1.1. Uždavinys | 3 |
| 1.1.2. Laboratorinis darbas 1 | 3 |
| 1.2. Kompiuterinės įrangos ir parametrų pasirinkimas | 4 |
| 1.3. Algortimų analizė | 4 |
| 1.3.1. Pure random search (PRS) lygiagretinimas | 4 |
| 1.3.2. Atstumų matricos skaičiavimo lygiagretinimas | 4 |
| 1.4. Rezultatų analizė | 5 |

1. 1 laboratorinis darbas

1.1. Aprašymas

1.1.1. Uždavinys

Aibę A sudaro geografiniai taškai, nurodant platumos ir ilgumos koordinates. Iš šios aibės reikia parinkti taškų aibę X tokią, kad atstumų nuo kiekvieno aibės A taško iki jam artimiausio aibės X taško suma būtų minimali $X \subset A$.

Faile `lab_data.dat` pateikiama 50000 geografinių taškų, kur viena eilutė aprašo vieno geografinio taško koordinates.

Faile `lab_01_2_algorithm.cpp` pateikiamas programos, kuri randa nurodyto n taškų aibę X , atitinkančią uždavinio sąlygą, naudojant paprastosios atsitiktinės paieškos (angl. Pure Random Search, PRS) algoritmą.

Pagrindiniai algoritmo parametrai (globalūs kintamieji):

- `num_points`: duomenų aibės A dydis (max 50000)
- `num_variables`: ieškomos taškų aibės X dydis
- `num_iterations`: sprendinio paieškai skirtų iteracijų skaičius (kuo daugiau, tuo didesnė tikimybė rasti geresnį sprendinį).

Algoritmo vykdymo pradžioje sudaroma atstumų matrica, kurioje saugomi atstumai kilometrais tarp taškų, suskaičiuoti pagal Haversino formulę. Atsižvelgiant į tai, kad atstumas nuo taško a iki taško b yra lygus atstumui nuo taško b iki taško a , yra užpildoma tik pusė matricos. Šioje matricoje saugomi atstumai yra naudojami vykdant aibės X taškų paiešką.

1.1.2. Laboratorinis darbas 1

1. Pasirinkti duomenų aibės dydį ir algoritmo iteracijų skaičių, kad atstumų matricos skaičiavimas užtruktų ne mažiau 10 sekundžių, o sprendinio paieškos laikas būtų nemažesnis nei 20 sekundžių.
2. Duomenų įkėlimą ir atstumų matricos skaičiavimą laikyti nuosekliąja algoritmo dalimi, o sprendinio paiešką - lygiagretinama dalimi, įvertinti teorinius galimus algoritmo pagreitėjimus naudojant 2 ir 4 procesorius, bei didžiausią galimą pagreitėjimą.
3. Duomenų įkėlimą ir atstumų matricos skaičiavimą laikyti nuosekliąja algoritmo dalimi, sudarykite lygiagretųjį bendros atminties algoritmą ir eksperimentiniu būdu ištirkite jo pagreitėjimą naudodami 2 ir 4 procesorius.
4. Sudarykite lygiagretų bendros atminties algoritmą atstumų matricos skaičiavimui ir eksperimentiniu būdu ištirkite jo pagreitėjimą naudodami 2 ir 4 procesorius.
5. Pananalizuoti, kai matricos reikšmių suskaičiavimą lygiagrečiaja dalimi, o pure random search (PRS), nuosekliąja.

1.2. Kompiuterinės įrangos ir parametrų pasirinkimas

Algoritmo analizei buvo naudojama **Apple Mac Mini Desktop Computer, 3.2GHz 6-Core Intel Core i7** kompiuteris, kurio dėka, buvo galima paleisti ant 2, 4 ir 6 procesorių. Kad įgyvendinti 1 nurodymą, buvo pasirinkta:

- num_points = 12000
- num_iterations = 30000

| Duomenų nuskaitymas (s) | Atstumų matricos skaičiavimas (s) | PRS skaičiavimas (s) |
|-------------------------|-----------------------------------|----------------------|
| 0.00323701 | 10.3124 | 19.9546 |
| 0.00437999 | 10.3154 | 19.993 |
| 0.00339818 | 10.3207 | 19.9673 |

1.3. Algoritmų analizė

1.3.1. Pure random search (PRS) lygiagretinimas

pakeisti i pseudo koda

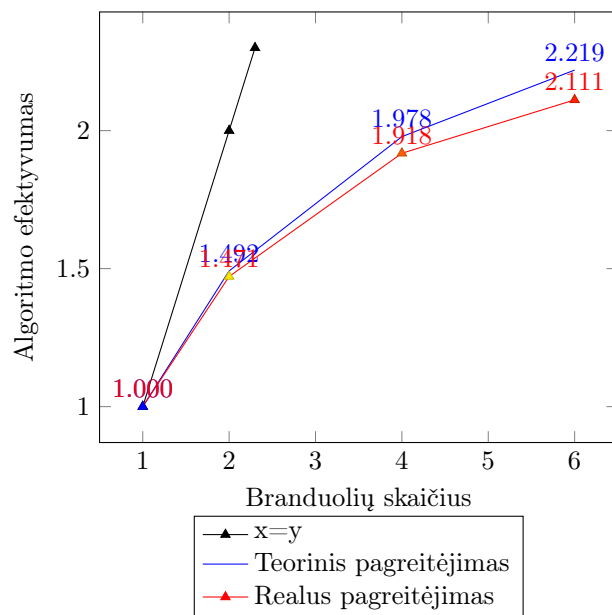
```
int *best_solution = new int[num_variables];
double f_solution, f_best_solution = 1e10;
#pragma omp parallel reduction (min: f_best_solution ) private (f_solution)
#pragma omp for schedule(dynamic)
for (int i=0; i<num_iterations; i++) {
    int *solution = new int[num_variables];
    random_solution(solution);
    f_solution = evaluate_solution(solution);
    if (f_solution < f_best_solution) {
        (mazesnis) uz geriausia zinoma
        f_best_solution = f_solution;
        if(f_best_solution == f_solution){
            #pragma omp critical (DataCollection)
            {
                for (int j=0; j<num_variables; j++) {
                    best_solution[j] = solution[j];
                }
            }
        }
    }
}
```

1.3.2. Atstumų matricos skaičiavimo lygiagretinimas

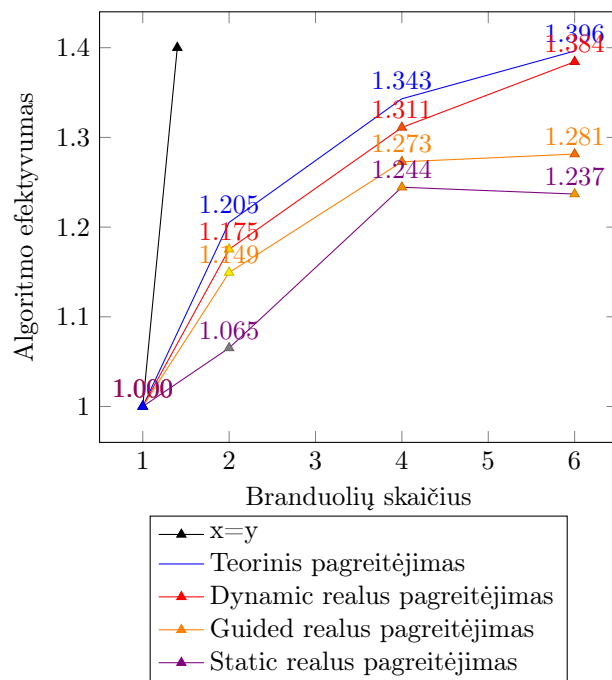
```
#pragma omp parallel for schedule(dynamic)
for (int i=0; i<num_points; i++) {
    distance_matrix[i] = new double[i+1];
    for (int j=0; j<=i; j++) {
        distance_matrix[i][j] = Haversine_distance(points[i][0], points[i][1], points[j][0], points[j][1]);
    }
}
```

1.4. Rezultatų analizė

PRS lygiagretinimas:



Matricos skaičiavimo laikai:



Abu sulygiagretinti:

