Generare arbori pentru evaluarea de binarizări optime

Raport Tehnic

1st Maria Ciuculan
Project Manager
Proiect MPS

2nd Ana-Maria Mihnea
Team Leader
Proiect MPS

3rd Andrei Cioban
Developer
Proiect MPS

4th Dragos Lucian Paune
Developer
Proiect MPS

5th Andrei Hirt
Tester
Proiect MPS

***Abstract*—Acest document descrie solutia prototipului intermediar, ce contine abordarea globala si sumarizeaza rezultatele software intermediare obtinute, arhitectura utilizata si concluziile obtinute din implementare.**

Keywords—arbori, optimizare, C++, procesare imagini, Scrum, software.

I. INTRODUCERE

Proiectul are la bază conceptul de binarizare globală și locală. Scopul proiectului este acela de a combina ieșirile unor algoritmi (praguri de binarizare - threshold) într-un mod inteligent pentru a obține un rezultat mult mai bun și mai general decât oricare din aceștia în mod individual.

Binarizarea este preprocesarea imagini ce presupune clasificarea pixelilor în două categorii: pixel de fundal și pixeli ce conțin obiectul (text sau alte obiecte), fiind o metodă de segmentare.

Thresholding sau nivel/valoare de prag este folosit pentru determinarea clasei unui pixel.

Binarizarea globală reprezintă aplicarea unui singur prag de binarizare pe întreaga imagine și transformarea pixelilor din cadrul acesteia în valori de alb (255) sau negru (0) în funcție de valoarea pragului.

II. DESCRIERE SOLUTIE

Se va utiliza conceptul de binarizare, prin doua metode, si anume cea de binarizare globala si cea locala, aplicandu-se asupra lor algoritmul Monte-Carlo, algoritm realizat in limbajul de programare C++.

Pentru realizarea abordarii binarizare globala s-a separat logica proiectului in diverse clase pentru o mai buna mentenabilitate si lizibilitate.

Algoritmul pentru generarea arborelui a fost implementat astfel incat sa se cunoasca numarul de frunze prestabilit(cele 15 threshold-uri).

S-a decis ca anumite frunze sa poata aparea de mai multe ori, pentru a acoperi cat mai multe combinatii de frunze in vederea generarii nodurilor intermediare. Astfel va fi o valoare N mai mare decat numarul de 15 frunze.

Pornirea algoritmului se face cu un vector de dimensiune N, ce va retine ordinea aparitiei frunzelor, fiind completat cu pozitii random din sirul de frunze. Pe baza acestor valori se vor stoca frunzele intr-o coada ce va retine nodurile ulterior generate.

Urmatorul pas este realizarea nuui loop, astfel cat timp N este mai mare decat 1, se alege random numarul de copii aferent unui nod (cuprins intre 2 si N) si tipul operatiei.

Aceasta pereche rezultata este stocata intr-o coada de alegeri iar N-ul este diminuat pe baza numarului de copii generat.

In final, cat timp coada de alegeri nu este epuizata, se scoate cate o pereche <operatie, nr_copii>, se creeaza nodul aferent pe baza perechii din coada (operatia este cea extrasa din pereche, iar numarul de copii (primele nr_copii) din coada de noduri, ce sunt dupa eliminate) si se adauga nodul nou creat in coada de noduri. In urma acestor operatii, in coada de noduri va ramana doar radacina ce va cunoaste intreaga topologie a arborelui.

III. ARHITECTURA

Tehnologiile utilizate pentru dezvoltarea proiectului sunt C++, Python, framework de testare unitara pentru C++: Google Test, GitHub. Organizarea proiectului s-a efectuat utilizand diagrama Gantt pentru planificare, Trello pentru structurare si gestionare si FlowChart pentru vizualizarea etapelor.

Ca metodologie de dezvoltare a proiectului s-a ales cea de tip Agile Scrum, fiind o metodologie de actualitate datorita faptului ca principiul acesteia este dezvoltarea incrementala a aplicatiei software, ulterior clientul primind livrari frecvente care contin un numar tot mai mare de functionalitati. De asemenea, pastrandu-se constant legatura cu clientul, se pot primi lamuriri

legate de specificatii si de modul in care acesta doreste sa primeasca produsul final.

Pentru realizarea abordarii binarizare globala s-a separat logica proiectului in diverse clase pentru o mai buna mentenabilitate si lizibilitate. Ierarhia de fisiere se prezinta astfel:

```
.
├── inputData
│   ├── inputData.cpp
│   └── inputData.h
├── node
│   ├── node.cpp
│   └── node.h
├── operation
│   ├── operation.cpp
│   └── operation.h
├── parser
│   ├── parser.cpp
│   └── parser.h
├── randomTree
│   ├── randomTree.cpp
│   └── randomTree.h
└── solution
    ├── Makefile
    └── main.cpp
```

IV. REZULTATE INTERMEDIARE ALE SOLUTIEI SOFTWARE

Etapa a presupus verificarea milestone-ului anterior si continuarea implementarii algoritmului. Astfel, se evidentiaza verificarea corectitudinii citirii din fisiere a datelor de intrare, verificarea corectitudinii parsării acestora si construirea corectă a arborelui.

V. CONCLUZII INTERMEDIARE

Milestone-ul 2 a presupus implementarea abordarii globale, indeplinindu-se astfel task-urile asignate.

Avand in vedere rationamentul pentru milestone-ul 2, se poate evidentia faptul ca primul milestone a fost realizat cu success, in timpul alocat in diagrama Gantt.

Dificultatea cea mai mare a fost intampinata in momentul construirii arborelui. Tinand cont ca exista drept input doar frunzele acestuia, pentru o implementare cat mai eficienta a solutiei, rationamentul ales a fost de a-l genera bottom-up.

Una dintre problemele intampinate a fost reprezentata de neconcordanta intre versiunile de compilator ale developer-ilor, anume in momentul citirii datelor din fisiere unul dintre developeri folosit o clasa ce nu era valabila pentru versiunile mai vechi de g++. Acest lucru a generat o latenta in procesul de dezvoltare.

VI. REFERINTE

[1] - https://www.overleaf.com/latex/templates/ieee-conference-template/grfzhhncsfqn

[2] - https://ieeexplore.ieee.org/document/8389021

[3] - https://ieeexplore.ieee.org/document/5709080