**UNIVERSITATEA DIN BUCUREȘTI**

**FACULTATEA**

**DE**

**MATEMATICĂ ȘI INFORMATICĂ**

**SPECIALIZAREA INFORMATICĂ**

**Lucrare de licență**

**DRUMURI DE LUNGIME MINIMĂ PE SUPRAFETE TRIANGULATE**

**Absolvent**

**Duncea Vlad-Alexandru**

**Coordonator ştiinţific**

**Lect. Dr. Stupariu Mihai-Sorin**

**București, iunie 2021**

**Rezumat**

# Cuprins

Cuprins 4

Introducere 5

I. Algoritmi pe muchii 6

1. Dijkstra 6

2. Dijkstra Bidirectional 6

3. A\* 6

4. Rafinare iterativa 6

I. Algoritmi pe fețe 6

# Introducere

# Algoritmi bazati pe grafuri

## Dijkstra

## Dijkstra Bidirectional

## A\*

## Rafinare iterativa

# Algoritmi pe fețe

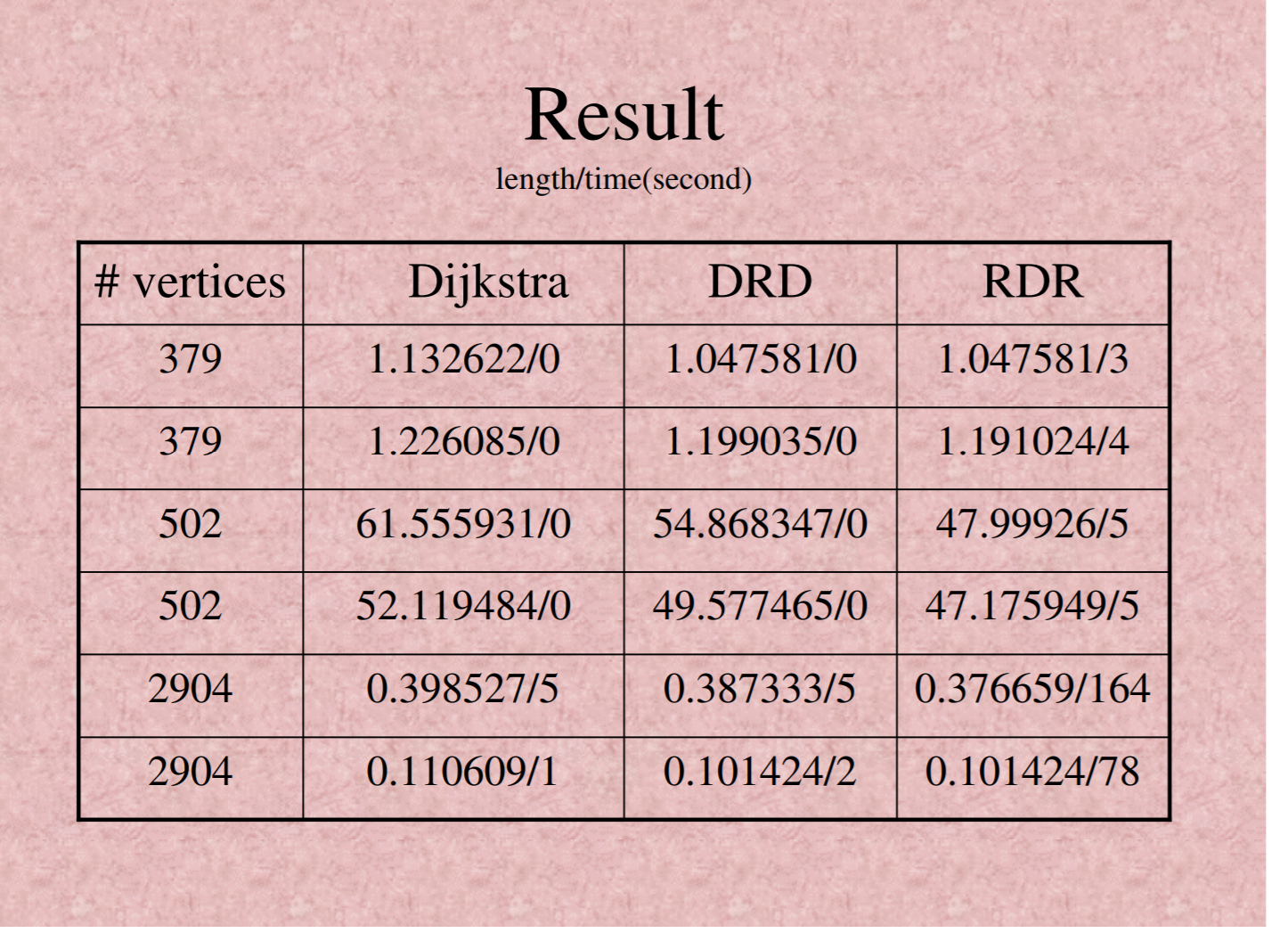
Conventii:

{n = numarul de noduri, m = numarul de muchii}

Cei trei algoritmi functioneaza atat pe suprafete traingulate, dar si pe orice forma geometrica formata din puncte si legaturi intre acestea. Vom observa ca algoritmii mai complecși care se bazeaza pe ‚intinderea’ suprafetelor sau pe rafinarea punctelor vor putea rula doar pe suprafete/obiecte triangulate, adica nu putem avea intersectii de muchii sau fete cu mai mult sau mai putin de 3 muchii.

1. Backtracking
   * Banal
   * Timp exponential de rulare (inutil pentru orice problema cu un numar mare de triunghiuri !de vazut timpi aproximativi pentru a putea spune cat este prea mult)
   * Genereaza cel mai scurt drum daca avem constrangerea de a merge pe muchii
2. Dijkstra
   * Algoritm simplu de implementat(complexitate mica)
   * Timp de rulare O(n^2) sau in cel mai bun caz(cu optimizari) O(mlogn)
   * In momentul in care extrage un nod nu se mai intoarce la el. Daca observam vizual parcurgerea algoritmului vom observa o propagare a nodurilor verificare incepand cu nodul de start (ca valurile facute de o picatura in apa). **De adaugat poza cu propagarea**
   * Gaseste la fiecare pas drumul minim de la punctul de start la punctul extras la iteratia curenta.
3. A\*
   * Algoritm simplu de implementat (complexitate medie)
   * Necesita o functie de aproximare. Avand in vedere ca noi suntem intr-un spatiu tridimensional Euclidian, putem cu usurinta sa folosim distanta euclidiana a doua puncte care va fi intotdeauna mai mic sau egal cu distanta pe suprafata noastra.
   * Spre deosebire de efectul de propagare al algoritmului Dijkstra, A\* are o propagare directionala, la fiecare pas incercand sa se apropie de nodul tinta. **De adaugat poza cu propagarea**
   * Pe suprafete mici unde nu poate folosi avantajul de a merge directionat, sau unde intalneste obstructii, A\* se va comporta mai prost decat Dijkstra. (De vazut unde apare aproximativ aceasta diferentiere)

**De facut un tabel comparativ(Backtracking VS Dijkstra VS A\*) cu distante,timpi de rulare, muchii verificate pe diferite forme triangulate**

**EX: **

**Idei:**

* Algoritm care ia drumul minim constrans la muchii, ia triunghiurile care au muchii in drum, construieste in mijlocul fiecarui triunghi un punct, genereaza noua triangulare(o vom numi rafinarea trangularii) si repeta. Va genera un drum de lungime aproximativa, care nu este constrans la muchiile initiale, insa nu va gasi drumul de lungime minima absoluta. Putem impune o limita de rafinare pentru avea un timp de rulare decent.
* Transformarea unei suprafete traingulate in spatiu 3D la o suprafata in spatiul 2D, acum, pentru a gasi drumul minim e suficient sa unim punctele, pentru ca ne aflam in un spatiu 2D. Complexitate foarte mare. Inca nu stiu cum sa fac asta. Alegem o secventa de triunghiuri si muchii, a.i. sa avem triungi,muchie,triunghi,muchie,... si fiecare muchie dintre doua fete este exact muchia comuna. Parcurgem lista si rotim dupa muche a.i. sa ajungem la coplanaritate, repetam pentru fiecare muchie pana cand toate triunghiurile sunt coplanare.
* De adaugat o pagina in care utilizatorul poate selecta fisiere noi pe care sa le adauge in aplicatie (.txt/.obj)
* De discutat la partea de alg cu rafinare de diferenta dintre rafinarea in 3 fete versus 6 fete.

Date de intrare:

* Fisiere txt cu urmatorul format: Pe prima linie se afla numarul de puncte, il vom numi *n*. Pe urmatoarele *n* linii se vor afla 3 numere in format intreg sau flotant. Pe urmatoarele *n* vom avea listele de adiacenta cu formatul: Doua numare, primul reprezentand nodul (primul nod este 0) pentru care urmeaza lista de vecini, al doilea reprezentand numarul de vecini ai nodului (notam cu *v*), apoi *v* numere reprezentand vecinii. Toate cele *n* noduri trebuie sa aiba o linie, daca un nod nu are vecini va aparea ca `index\_nod 0`.
* Vreau sa pot introduce in aplicatie si fisiere de tipul *.obj*. Fisierele au un format simplu de parcurs, folosind primul caracter al fiecarei linii pentru a specifica ce va urma pe aceasta. De exemplu:
  + Liniile care incep cu ‚#’ sunt comentarii si vor fi ignorate
  + Liniile care incep cu ‚v’ sunt puncte geometrice cu minim 3 coordonate in ordinea x,y,z. A patra coordonata este optionala.
  + Liniile care incep cu ‚f’ sunt urmate de indicii punctelor care formeaza o fata. Exista mai multe metode prin care pot fi date, de ex se pot specifica si coordonate de textura sau normale, insa in scopul aplicatiei acestea vor fi ignorate.
  + In standard mai exista ‚l’ pentru linii, ‚vt’ pentru coordonate de texturi, ‚vn’ pentru coordonatele normalelor, ‚vp’ pentru puncte in spatiu parametrizat(de ex multimea punctelor de pe un plan sau o dreapta) si multe altele.
  + Aplicatia va prelucra doar liniile incepand cu ‚v’ si ‚f’ pentru a putea construi suprafata, fie ea triangulata sau nu.

Utilizarea aplicatiei:

* Taste utilizate:
  + Tastele W, S deplaseaza camera in fata, respectiv in spate.
  + Tastele A, D deplaseaza camera la stanga, respectiv la dreapta.
  + Tastele Spatiu, C deplaseaza camera in sus, respectiv in jos.
  + La apasarea tastei Shift, Cursorul dispare si utilizatorul poate orienta camera folosind miscari ale Mouse-ului. Prin reapasarea tastei Shift cursorul va aparea si Mouse-ul nu va mai influenta orientarea camerei.

**Gasite in articole**

Dijkstra proposed an algorithm to solve the SSSP problem on a directed weighted graph G(V , E) with n vertices, m edges, and positive weights. Dijkstra’s algorithm proceeds by building a list of processed vertices for which the shortest path to the source point s is known. The algorithm iteratively decreases estimates on the shortest paths of non-processed vertices, which are stored in a priority queue. In each iteration of the algorithm, the closest unprocessed vertex from the source is extracted from the priority queue and processed by relaxing all its incident edges. The notion of relaxation underlines the analogy between the length of the shortest path and the length of an extended tension spring. When the algorithm starts, the length of the shortest path is overestimated and can be compared to an extended spring. In each iteration, a shorter path is found, which can be compared to relaxing the spring. Although the original implementation used O(n2) time, the running time was decreased to O(n logn + m) by using Fibonacci heaps [21]. Thorup [61] presented an O(m)-time algorithm for the case where each edge is assigned a positive integer weight. The algorithm runs on a RAM model and assumes that all weights and distances fit in a word each. The main idea is to use a hierarchical bucketing structure to avoid the bottleneck caused by sorting the vertices in increasing order from s.

[A survey of geodesic paths on 3D surfaces]

Because we can triangulate every face of the terrain in linear time [3] we may assume without loss of generality that every face of F is a triangle(Trekking in the Alps Without Freezing or Getting Tired1 M. de Berg2 and M. van Kreveld2)

# Concluzii

# Bibliografie

**There are no sources in the current document.**