FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ,

UNIVERSITATEA DIN BUCUREȘTI

DOMENIUL DE LICENȚĂ: INFORMATICĂ

PROGRAMUL DE STUDIU: INFORMATICĂ

**LUCRARE DE LICENȚĂ**

Aplicații ale algoritmilor euristici în parcurgeri de drumuri minime

ÎNDRUMĂTOR ȘTIINȚIFIC

***Lect. Dr. Marinescu-Ghemeci Ruxandra***

ABSOLVENT

***Munteanu Adrian***

București

2015

Cuprins

[Introducere 3](#_Toc419881376)

[Capitolul I. HTML5, ECMAScript 6 și WebGL 4](#_Toc419881377)

[I.1. HTML5 4](#_Toc419881378)

[I.2. ECMAScript 6 4](#_Toc419881379)

[I.3. WebGL 4](#_Toc419881380)

[I.4. Three.JS 5](#_Toc419881381)

[I.5. Browserul Google Chrome 5](#_Toc419881382)

[Capitolul II. Modelul matematic al problemei și soluțiilor 6](#_Toc419881383)

[II.1. Ipoteza problemei 6](#_Toc419881384)

[II.2. Soluții 6](#_Toc419881385)

[II.2.1. Căutarea în lățime 7](#_Toc419881386)

[II.2.2. A\* 7](#_Toc419881387)

[II.2.3. D\* 7](#_Toc419881388)

[II.2.4. HGA\* 8](#_Toc419881389)

[II.2.5. A\* ponderat (Weighted A\*) 8](#_Toc419881390)

[II.2.6. Jump Point Search (JPS) 8](#_Toc419881391)

[II.3. Structuri de date necesare 9](#_Toc419881392)

[Capitolul III. Aplicație 10](#_Toc419881393)

[III.1. Librării și instrumente 10](#_Toc419881394)

[III.2. Detalii de implementare 10](#_Toc419881395)

[III.3. Testare 10](#_Toc419881396)

[III.4. Rezultate 10](#_Toc419881397)

# Introducere

Eficiența și optimizarea algoritmilor reprezintă o continuă cercetare în domeniul informaticii. Aceste îmbunătățiri implementate atât hardware cât și software pot aduce o diferență majoră în domeniul roboticii.

Scopul acestei lucrări este de a expune vizual comparația diferiților algoritmi utilizați în programarea roboților pentru a se deplasa într-un teren parțial cunoscut.

# Capitolul I. HTML5, ECMAScript 6 și WebGL

Începând cu anul 2011, când a fost propusă o variantă finală pentru HTML5 de către grupul W3C, s-a putut observa o mișcare în vederea standardizării a tot mai multe funcționalități pentru navigatoare de internet. De asemenea, aceste noi funcționalități sunt însoțite de îmbunătățiri la nivelul browser script.

Tot în anul 2011 a fost lansată versiunea 5.1 a ECMAScript, venind ca o completare la HTML5. WebGL, sau Web Graphics Library, a fost de asemenea introdus, iar la câteva luni avea sa apară librăria Three.js pentru a facilita dezvoltarea aplicațiilor 3D care rulează direct în browser.

## I.1. HTML5

Pachetul de îmbunătățiri cu care vine HTML5 conține noul element **canvas**. Acest element reprezintă o porțiune dreptunghiulară în pagina web unde poate se poate „desena”. Acest procedeu de a desena poate fi realizat folosind funcționalități noi de programare din JavaScript ce permit accesarea conținutului zonei de canvas și modificarea acestuia.

## I.2. ECMAScript 6

ECMAScript este un limbaj de scripting standardizat și este folosit la scară largă pentru scripturi la nivel de client de aplicație. El stă la baza limbajului JavaScript, ceea ce face compatibilă utilizarea lui în diferite implementări de browser web.

Versiunea 6 a ECMAScript aduce ca elemente de noutate, printre altele, structuri noi de date, cum ar fi **Set** (mulțime de elemente) și **Map** (hartă cheie/valoare), și **iteratori** și **for…of**. Având aceste instrumente la dispoziție este mai ușor sa menținem, de exemplu, lista de vecini ai unui nod.

Exemplu:

|  |  |  |
| --- | --- | --- |
| ECMAScript 5 | ECMAScript 6 |  |
| var obj = {};  obj[x] = true;  delete obj[x];  if(x in obj){ … } | var obj = new Set();  obj.add(x);  obj.delete(x);  if(obj.has(x)){ … } | //se creaza un obiect  //se seteaza o valoare  //se sterge o valoare  //se verifica existenta unei valori |
|  |  |  |

## I.3. WebGL

WebGL este o specificație de JavaScript menită să îmbunătățească experiența de navigare pe web venind cu grafică 3D și 2D la dispoziția browser-ului web fără a fi nevoie de vreun program adițional. Astfel, conținutul unui site web poate sa folosească placa video a calculatorului pentru cea mai bună performanță. Această librărie este bazată pe librăria multi-platformă OpenGL.

## I.4. Three.JS

Three.JS este o librărie scrisă în JavaScript ce ajută la implementarea aplicațiilor 3D și oferă, dar nu se rezumă la următoarele facilități:

* Motoare de randare grafica: WebGL, <canvas>, <svg>, CSS3D, DOM, Software
* Scene: pentru a adăuga și elimina obiecte în timpul rulării
* Camere: de perspectivă și ortografică
* Animații
* Lumini: de ambient, direcționale, punctiforme; umbre
* Materiale: Lambert, Phong, cu texturi și umbrire netedă
* Obiecte: rețea, particule, sprites, lumini
* Geometrii: plan, cub, sferă, 3D text; modificatori: alungire, extrudare și tăiere
* Funcții matematice cum ar fi manipulări de matrice, cuaternioni, UV

Ilustrarea unei simulări 3D poate fi realizată direct pe orice navigator compatibil și poate fi la fel de performantă ca o aplicație ce rulează doar pe anumite sisteme de operare sau dispozitive. De exemplu, pentru a crea spațiul geometric necesar va fi nevoie doar de o geometrie de tip plan, având un material și opțional o textură. Având acces direct în JavaScript la aceste facilități putem apoi folosi evenimentele de mouse și tastatură pentru a manipula scena și obiectele din ea.

var geometry = new THREE.PlaneGeometry(width, height, widthSegments, heightSegments);

var material = new THREE.MeshBasicMaterial( {color: 0xffff00} );

var plane = new THREE.Mesh( geometry, material );

scene.add( plane );

## I.5. Browserul Google Chrome

Pentru a putea beneficia de cele mai noi tehnologii în domeniul web se poate folosi browserul Google Chrome, ce are implementate multe dintre funcționalitățile încă în proces de standardizare. Acesta suportă HTML5 și WebGL și o bună parte din ECMAScript 6.

# Capitolul II. Modelul matematic al problemei și soluțiilor

Problema studiată în acestă aplicație este găsirea celui mai scurt drum între două puncte în teren parțial cunoscut. Această problemă este echivalentă în practică în domeniul roboticii cu problema automatizării deplasării unui robot spre un punct destinație.

## II.1. Ipoteza problemei

Fie un robot cu instrumente pentru a primi următoarele date: poziția curentă, poziția destinație și mulțimea de muchii aflate în spațiul vizibil al robotului.

Definim în continuare harta pe care se află robotul ca o mulțime de puncte în spațiul geometric euclidian tridimensional, astfel un punct având cele 3 coordonate: x, ,y, z. Un **obstacol** este tradus ca o muchie de cost ∞.

Spunem că o muchie se află în spațiul vizibil al robotului dacă ambele capete se află la o distanță mai mică sau egală cu o **rază de viziune** fixată. Distanța dintre două puncte **p** și **q** este distanța euclidiană .

În figura *Fig.1* este marcată poziția curentă a doi roboți **(1)**, sfera de viziune **(2)**, mulțimea de muchii din spațiul vizibil **(3)**, marcate cu albastru, și poziția destinație **(4)**.

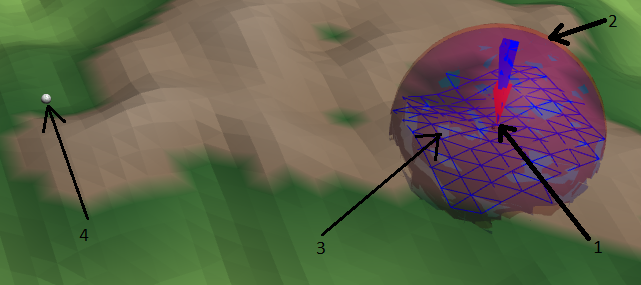


Fig.1. Ilustrare

Se poate observa faptul că în afara sferei de viziune a roboților nu este marcată nicio muchie, deoarece terenul nu este cunoscut. Din această cauză algoritmii folosiți trebuie să gasească cel mai scurt drum ținând cont de incompletitudinea informației.

## II.2. Soluții

Pentru a rezolva problema celui mai scurt drum putem implementa diverși algoritmi, cum ar fi căutarea în lățime, Dijkstra, A\*, D\* și altele. Însă pentru a putea aplica acești algoritmi este necesar să existe cel puțin un lanț între cele două puncte, ceea ce nu se întâmplă frecvent pentru configurația aleasă. De aceea, introducem noțiunea de **nod vizibil complet**, desemnând un nod pentru care robotul despre care vorbim nu cunoaște toți vecinii. Simplificăm și mai mult situația în continuare fixând lungimea maximă a unei muchii cu o valoare strict mai mică decât raza de viziune a robotului (roboților).

Combinând aceste modificări, observăm că un **nod vizibil complet** este unul care satisface condiția , unde *viz*este raza de viziune a robotului, *R* este poziția curentă a robotului și *n* este poziția nodului. Spunem astfel că un nod este **vizibil parțial** dacă nu este vizibil complet.

Astfel, pentru cazul în care nu există un drum în graful cunoscut al robotului este suficient să căutăm cel mai scurt drum spre nodul vizibil parțial cu cea mai mică **distanță estimată**. De asemenea, se poate întampla și ca „drumul cunoscut sa *nu* fie cel mai scurt”, existând posibilitatea ca unele noduri vizibile parțial să conducă pe un drum mai scurt.

Definim în continuare **distanța estimată** *h(x) = d(x, s)*, unde *s* este poziția destinație. Această distanță este astfel o euristică admisibilă deoarece îndeplinește condiția pentru orice *x* și *y* noduri adiacente.

### II.2.1. Căutarea în lățime

Având în vedere faptul că robotul se află într-un teren necunoscut, această căutare va fi efectuată după fiecare actualizare a cunoștințelor robotului. Astfel, la primul traseu parcurs, algoritmul va căuta un drum minim înainte de fiecare mișcare.

Este cunoscut faptul că această parcurgere are complexitatea timp , unde *V* este mulțimea vârfurilor și *E* este mulțimea muchiilor.

Pentru a îmbunătăți acest algoritm ne putem folosi de distanța estimată introdusă pentru a alege vârful intermediar și o putem folosi și pentru a alege vârfurile explorate.

### II.2.2. A\*

Deși este în continuare necesar să efectuăm o căutare completă la orice actualizare, de această dată algoritmul A\* va explora semnificativ mai puține noduri. Acest efect este dat de faptul că algoritmul A\* prioritizează nodurile care se „apropie” de soluție, în sensul că deși costul de a ajunge în aceste noduri este mai mare, distanța estimată este mai mică.

Deoarece euristica *h* trebuie să fie admisibilă pentru ca algoritmul să determine soluția optimă acest algoritm are și dezavantaje. Pentru cazul în care destinația se află în spatele unui obstacol acest algoritm se aseamănă cu căutarea în lățime, având nevoie de timp și memorie. Cazul cel mai nefavorabil dă algoritmului o complexitate , ceea ce în aplicații practice poate constitui o problemă.

### II.2.3. D\*

Pornind de la algoritmul A\* putem optimiza rularea lui folosindu-ne de drumul minim găsit la un anumit moment dat și adaptându-l conform noilor informații. Astfel, algoritmul D\* introduce notația „right-hand side value” – valoarea optimă pentru un vârf.

unde *Pred(s)* este mulțimea vârfurilor predecesori lui s

Spunem că un vârf *s* este **consistent local** daca *g(s) = rhs(s)*, altfel este **inconsistent local**. Dacă toate vârfurile sunt consistente local atunci valoarea *g* a lor este egală cu distanța respectivă de la start, ceea ce permite găsirea drumului minim de la start la orice vârf. Totuși, vom folosi euristica *h* pentru a focusa pe destinație. Se va folosi astfel o coadă de priorități ce va reține exact vârfurile inconsistente local.

Cheia pentru coada de priorități o formăm din două valori: , unde și , comparația cheilor facându-se în ordine lexicografică. Datorită alegerii cheii în acest fel, prioritatea nodurilor din coadă va fi dată de cel mai mic cost pentru a ajunge în nod și euristica din acel nod, spărgând egalitățile in favoarea costului cel mai mic spre un nod.

### II.2.4. HGA\*

Un alt algoritm pentru determinarea unui drum între două puncte este HGA\*. Acesta însă construiește o soluție aproximată pentru a reduce timpul de calcul. Idea de bază a algoritmului este găsirea de rute ocolitoare pentru obstacolele aflate pe traiectoria de mers, traiectoria inițială fiind între punctul de start și cel de sosire. Astfel, algoritmul caută cea mai apropiată cale liberă pornind de la intersecția cu primul obstacol și se construiesc recursiv astfel rute ocolitoare ce pot fi ordonate după cost.

### II.2.5. A\* ponderat (Weighted A\*)

A\* ponderat aduce o modificare minoră în schema algoritmului A\* introducând o relaxare a criteriului de admisibilitate. Funcția euristică *h* se va înmulți cu ce reprezintă înclinarea către nodurile mai apropiate de destinație, deoarece funcția pe baza căreia se face comparația nodurilor devine .

Din cauza ușoarei modifcări a ordinei de explorare acest algoritm nu găsește soluția optimă, însă este demonstrat faptul că soluția găsită va avea costul mai mare decât soluția optimă de maxim ori, ceea ce în practică se dovedește de multe ori a fii un compromis acceptabil.

### II.2.6. Jump Point Search (JPS)

Jump Point Search este o altă metodă de a optimiza algoritmul A\* mai ales pentru căutarea într-o matrice. Prin acest procedeu sunt eliminați vecinii candidați în care se poate ajunge din pasul anterior, determinând astfel puncte prin care trebuie sa treacă drumul minim, zise puncte de salt (jump points). Această metodă găsește soluția optimă reducând timpul de rulare cu un ordin de magnitudine.

## II.3. Structuri de date necesare

Eficiența structurilor de date folosite stă la baza algoritmilor eficienți. În cazul de față coada de priorități este una dintre ele. O implementare simplistă și ineficientă este, de exemplu, utilizarea unui vector, iar la fiecare extragere de minim se va face o parcurgere. Această metodă are o complexitate de .

Printre metodele folosite de obicei se află Binary Heap, fiind un arbore binar complet care satisface proprietatea ca orice nod să fie mai mic sau egal decât copiii lui. Complexitatea operațiilor este pentru inserare, extragere de minim și modificare de prioritate.

Fibonacci Heap este o altă implementare a cozii de priorități, ce are la bază o listă de arbori. Această structură nu se consolidează la inserare, ci doar la extragere. Principiul este de a construi arbori heap de adâncime crescătoare, spre exemplu, pentru n noduri vor exista maxim log(n) arbori în listă, iar arborele va avea maxim noduri dispuse în sub-arbori de adâncimi de la 0 la i-1, pentru orice . Această structură este menținută recursiv. Operația de inserare va presupune simpla adăugare a nodului nou în lista de arbori, reprezentând o rădăcină. În acest caz operația este executată în timp constant – – lăsând reordonarea nodurilor pentru alte operații. Această abordare nu influențează rezultatul, deoarece la extragerea minimului se consolidează structura, operația executându-se în timp amortizat reprezentând adâncimea unui arbore oarecare din listă.

# Capitolul III. Aplicație

…

## III.1. Librării și instrumente

jQuery, Three.JS, extensii ThreeX, editor WebStorm, versionare git cu repository pe GitHub…

## III.2. Detalii de implementare

Graf, plan, expunere, interacțiune…

## III.3. Testare

Scenarii de testare, ușurința utilizării, resursele necesare pentru utilizare

## III.4. Rezultate

Rulare – numar de pași efectuați, distanța parcursă, dimensiunea grafului cunoscut, timp și spațiu de rulare. Statistici – corelații, complexitate, aplicabilitate.