Proiect Analiza Algoritmilor: Drumuri minime in graf

Selea Tudor Octavian
322CA
selea.tudor.00@gmail.com
Etapa finală

Facultatea de Automatică și Calculatoare
Universitatea Politehnica din Bucuresti

1 Introduction

1.1 Descrierea problemei rezolvate

O problemă de drum minim include un graf orientat $G = (V, E)$, care are asociată o funcție de cost $w : E- > R$. Funcția de cost atribuie fiecărei muchii un cost reprezentat de un număr real. Costul unui drum de la nodul s la nodul v este reprezentat de suma costurilor tuturor muchiilor care alcătuiesc drumul. "Costul minim al drumului dintre două noduri este minimul dintre costurile drumurilor existente între cele două noduri" [2]. Scopul problemei alese este să determine costul minim al drumului dintre nodul sursă și toate celelalte noduri din graf. Domeniul funcției de cost poate fi extins, pentru a include si perechile de noduri care nu sunt legate prin niciun drum (din nodul s nu pot să ajung în nodul v), caz în care costul drumului dintre cele 2 noduri este $+\infty$. Trebuie precizat și faptul că pot exista muchii care au asociate costuri negative într-un graf. Dacă graful G nu conține cicluri cu cost negativ, accesibile din nodul s, atunci costul minim dintre nodul sursă și nodul destinație este bine-definit, oricare ar fi nodul destinație. În caz contrar, costul minim nu este bine-definit(dacă încercăm să calculăm drumul minim într-un graf cu cicluri de cost negativ, lungimea acestuia o să fie infinită întrucât costul scade la fiecare parcurgere a ciclului; din această cauză, costul drumului este $-\infty$).

1.2 Exemple de aplicații practice pentru problema aleasă

Un exemplu foarte bun de aplicație practică pentru distanța minimă de la un nod sursă la celelalte noduri din graf este determinarea celui mai scurt drum dintre 2 orașe. Aplicații precum Waze sau Google Maps transformă hărți rutiere (cum ar fi harta rutieră a României) în grafuri în care nodurile sunt orașe, muchiile sunt drumurile care leagă orașele, iar costurile muchiilor sunt distanțele dintre orase, și calculează costul minim dintre 2 noduri date ca input. Algoritmii pentru

determinarea costului minim al drumului dintre un nod sursă și celelalte noduri au aplicații practice și în:

- rețele de socializare (Facebook, Instagram)
- rețeaua de fibră optică folosită pentru traficul de date în internet
- rețeaua de transport a energiei electrice folosită pentru alimentarea cu energie electrică a consumatorilor casnici și a celor industriali
- rețeaua de linii de autobuz ce face parte din sistemul public de transport al unui oraș

1.3 Specificarea soluțiilor alese

Algoritmii aleși pentru soluționarea problemei drumului minim de la un nod sursă la toate celelalte noduri sunt:

- Dijkstra
- Bellman-Ford
- Cea mai eficientă metodă pentru grafuri orientate aciclice

1.4 Specificarea criteriilor de evaluare alese pentru validarea soluțiilor alese

Pentru evaluarea corectitudinii soluțiilor alese, am generat manual 7 teste bazate pe 3 grafuri, preluate de pe internet, cu un număr de noduri mic, cuprins între 5 și 9, pentru a le putea verifica atât manual (pe foaie), cât și folosind un utilitar care calculează distanța minimă de la un nod la toate celelalte.

Pentru evaluarea eficienței soluțiilor alese, am implementat un algoritm care generează liste de adiacență pentru grafuri. Pentru fiecare graf, numărul de noduri este cuprins între 20 și 1500, iar costurile muchiilor sunt generate random si au valori cuprinse intre 0 si 4999. Nu am generat teste care conțin grafuri cu mai mult de 1500 de noduri deoarece timpul de rulare al algoritmului Bellman-Ford era prea mare. Testele generate conțin grafuri cu 0 muchii, grafuri dense, grafuri rare și grafuri "random" (nici dese, nici rare).

Trebuie precizat faptul ca toate grafurile din testele generate (atât cele generate manual, cât și cele generate folosind algoritmul precizat mai sus) conțin doar muchii de cost pozitiv și sunt orientate aciclice. Am ales să generez testele astfel ca să pot aplica fiecare test pe toți cei 3 algoritmi (algoritmul Djikstra nu funcționează corect pentru grafuri cu muchii de cost negativ, iar Cea mai eficientă metodă pentru grafuri orientate aciclice nu funcționează pentru grafuri care conțin ciclii).

2 Prezentarea soluțiilor

2.1 Algoritmul Dijkstra

Algoritmul Dijkstra determină cel mai scurt drum de la un nod sursă la toate celelalte noduri și poate fi aplicat numai pe grafuri orientate care nu au muchii cu cost negativ. "Algoritmul este de tip Greedy: optimul local căutat este reprezentat de costul drumului dintre nodul sursă s și un nod v" [3]. Algoritmul se bazează pe principiul relaxării, care presupune obținerea de valori din ce în ce mai mici ale distanței dintre nodul sursă și nodul destinație, până când costul minim al drumului dintre cele 2 noduri este atins.

Modul de funcționare al algoritmului

- sunt folosite 2 structuri:
  - o mulțime de noduri care, inițial, o să conțină doar nodul sursă, și în care o să adăugăm nodurile care au cea mai mică distanță până la sursă calculată
  - o coadă de priorități în care se vor afla nodurile neprocesate în ordinea distanței față de sursă

- pentru fiecare nod se va reține un cost estimativ (inițializat cu ∞; nodul sursă este inițializat cu 0)
- se extrage un nod din coada de priorități și se relaxează toate muchiile care fac legătura dintre nodul extras și vecinii acestuia (fie u nodul extras, v nodul vecin și s nodul sursă. Dacă $d[s,v] > d[s,u] + d[u,v]$, atunci $d[s,v] = d[s,u]+d[u,v]$). Procedeul continuă până când coada de priorități este golită.

Analiza complexității

Construirea cozii de priorități durează $O(|V|)$ pentru $|V|$ noduri. Deoarece grafurile sunt reprezentate sub forma unor liste de adiacență, nodurile grafurilor pot fi parcurse folosind BFS, astfel ca parcurgerea vecinilor unui nod și actualizarea costurilor acestora au o complexitate de $O(|E|)$. Parcurgerea fiecărui nod din coada de priorități are o complexitate de $O(|V|)$, iar extragerea elementului minim din coada de priorități (implementată ca un minheap) și reordonarea acestuia după un update al costului unui nod au o complexitate de $O(log|V|)$. În final, complexitatea algoritmului este $O(|V|)+O(|E|*log|V|)+O(|V|*log|V|) = O((|E|+|V|)*log|V|) = O(|E|*log|V|)$, unde V = numărul de noduri, iar E = numărul de muchii [13]. O implementare cu o complexitate mai bună folosește un heap Fibonacci pentru coada de priorități ($O(|E|+|V|*log|V|)$, foarte bună pentru grafuri rare). Complexitatea algoritmului este $O(|V|^2)$ dacă graful este reprezentat ca matrice de adiacență, iar coada de priorități este reprezentată ca o listă normală.

Avantaje și dezavantaje

- Avantaje:
  - Este eficient dacă folosește un minheap($O(|E|*log|V|)$) sau heap Fibonacci($O(|E|+|V|*log|V|)$) pentru coada de priorități, iar graful este reprezentat ca o listă de adiacență
  - Are o aplicabilitate foarte mare (folosit de Google Maps pentru calcularea distanței minime dintre 2 orașe, în rețele de socializare, în IP routing)
  - Este garantat că va găsi cel mai scurt drum de la un nod la toate celelalte noduri (spre deosebire de alte euristici [13])

- Dezavantaje:
  - Nu poate calcula corect distanța minimă de la un nod sursă la toate celelalte noduri într-un graf cu muchii de cost negativ. Acest lucru se datorează faptului că algoritmul este de tip greedy, astfel că după marcarea unui nod ca vizitat (după ce este scos din coada de priorități), nu se mai poate reveni la el, chiar daca există un drum cu cost mai mic până la acesta. Această problemă apare numai dacă există o muchie de cost negativ în graf
  - Caută "în orb" drumul minim, astfel că există destul de mult timp irosit.

2.2 Bellman-Ford

Algoritmul Bellman-Ford oferă tot o soluție pentru problema drumului minim de la un nod sursă la toate celelalte noduri. Deși este mai lent decât algoritmul Dijkstra, algoritmul Bellman-Ford este mai versatil întrucât poate fi folosit și pentru grafuri care au muchii cu cost negativ. Algoritmul nu poate fi folosit pentru grafuri care contin cicluri de cost negativ, accesibile din nodul sursă. În schimb, poate semnala existența unor astfel de cicluri (printr- o valoare booleană).

Modul de funcționare al algoritmului:

- pentru fiecare nod se va reține un cost estimativ (inițializat cu ∞; nodul sursă este inițializat cu 0)
- se va parcurge fiecare nod din graf, iar la fiecare iterație se va încerca relaxarea fiecărei muchii din graf (relaxarea decurge la fel ca în cazul algorimului Dijkstra). Întrucât drumul minim dintre nodul sursă și orice alt nod din graf poate să treacă prin maxim $|V|$ noduri, este suficent ca fiecare muchie să fie relaxată de $|V|$ ori (unde V este numărul de noduri din graf) pentru a determina distanțele minime dintre nodul sursă și toate celelalte noduri (de aici vine și complexitatea algoritmului).
- dacă la finalul relaxărilor tuturor muchiilor de $|V|$ ori mai există o muchie care poate fi relaxată, înseamnă că graful conține un ciclu de cost negativ, iar problema nu are soluție.

Analiza complexității
Complexitatea parcurgerii fiecărui nod din graf este $O(|V| - 1)$. Cum pentru fiecare nod parcurgem toate muchiile din graf (și le actualizăm dacă este cazul), complexitatea algoritmului este $O(|E| * |V|)$.

Avantaje și dezavantaje

- Avantaje:
  - Este un algoritm dinamic
  - Funcționează și pentru grafuri cu muchii de cost negativ
  - Detectează cicluri de cost negativ
  - Funcționează bine pentru sisteme distribuite

- Dezavantaje:
  - Funcționează doar pentru grafuri orientate cu muchii de cost negativ
  - Este destul de lent

2.3 Cea mai eficientă metodă pentru grafurile orientate aciclice

Acest algoritm funcționează pentru grafurile orientate aciclice și se bazează pe **sortare topologică**. Toate drumurile minime într-un astfel de graf sunt bine definite deoarece nu pot exista cicluri de cost negativ (chiar dacă pot exista muchii de cost negativ).

Modul de funcționare al algoritmului:

- la fel ca în cazul algoritmilor Bellman-Ford și Dijkstra, pentru fiecare nod se va reține un cost estimativ (inițializat cu ∞; nodul sursă este inițializat cu 0)
- se va sorta topologic graful, iar nodurile sortate topologic se adaugă într-o stivă S
- după sortare, se va parcurge fiecare nod al grafului (se dă pop la stivă pentru a parcurge nodurile în ordine topologică) și se vor relaxa toate muchiile divergente din acel nod

Analiza complexității
Întrucât sortarea topologică se bazează pe DFS, complexitatea acesteia este $O(|E| + |V|)$. Apoi, pentru fiecare nod, muchiile divergente din acesta sunt parcurse exact o dată. Deducem, astfel, că parcurgerea nodurilor în ordine topologică și a muchiilor divergente din acestea are o complexitate de $O(|E| + |V|)$. În final, complexitatea algoritmului este de $O(|E| + |V|)$.

Avantaje și dezavantaje

- Avantaje:
  - Are cea mai bună complexitate pentru grafuri orientate aciclice dintre toți algoritmii prezentați

- Dezavantaje:
  - Nu poate fi folosit în grafuri care conțin ciclii datorită sortării topologice (dacă graful ar conține un ciclu, nu s-ar putea stabili o ordine între nodurile care fac parte din ciclu, deci sortarea topologică nu ar putea genera o ordine corectă a nodurilor)

3 Evaluare

3.1 Descrierea modalității de construire a setului de teste

Pentru testarea corectitudinii și eficienței algoritmilor am generat 41 de teste. Primele 7 teste le-am generat manual pentru a verifica corectitudinea algoritmilor:

- testele 1-3 conțin un graf preluat de pe OCW (am atribuit muchiilor costuri random) și le-am verificat atât pe foaie, cât și pe acest utilitar [5]
- testul 4 conține un graf preluat de aici [6]
- testul 5 conține un graf preluat de aici [7]
- testele 6-7 conțin un graf preluat de pe Wikipedia [8]
  Testele 4-7 au fost, de asemenea, testate pe site-ul precizat în descrierea testelor 1-3.

Următoarele 34 de teste le-am generat folosind un program scris in C. Acesta generează, mai întâi, o matrice de adiacență pe care o scrie într-un fișier de input (în folderul "in") sub forma unei liste de adiacență în felul următor:

1. adaugăm pe prima linie din fisierul de input numărul de noduri, numărul de muchii și nodul sursă
2. parcurgem fiecare element al matricei de adiacență
3. daca elementul este 0, trecem mai departe
4. daca elementul este diferit de 0, scriem în fisierul de input, pe o nouă linie, linia din matricea de adicență la care se află elementul (aceasta va reprezenta nodul-sursă), coloana din matricea de adiacență la care se află elementul (aceasta va reprezenta nodul-destinație) și valoarea elementului (aceasta va reprezenta valoarea muchiei dintre nodul-sursă și nodul-destinație)

- Testele 8 - 12 conțin grafuri cu 0 muchii și cu un număr de noduri cuprins între 50 și 1000

– Testele 13 - 21 conțin grafuri rare (”gradul de raritate” este dat de linia de cod ”$if(rand()\%15 == 0)$”. Pentru a face graful și mai rar, se poate înlocui 15 cu un număr mai mare). Numărul de noduri este cuprins între 20 și 1500
– Testele 22 - 30 conțin grafuri dense, cu un număr de noduri cuprins între 20 și 1500 de noduri
– Testele 31 - 41 conțin grafuri random(mai dese sau mai rare, în funcție de rezultatul lui $rand()\%5000$)

Trebuie precizat faptul că, întrucât grafurile generate trebuie să fie aciclice, matricile folosite pentru obținerea listelor de adiacență sunt superior triunghiulare (faptul că matricile sunt superior triunghiulare ne garantează faptul că grafurile generate cu ajutorul acestora sunt aciclice). Grafurile ”dense” sunt, de fapt, matrice care au elementele de deasupra coloanei principale diferite de 0. Aceste matrice ne generează grafuri cu $O((n * (n-1))/2)$ muchii (unde n este numărul de noduri), acesta fiind numărul maxim de muchii pe care îl poate avea un graf orientat aciclic.

3.2 Specificațiile sistemului de calcul

Mai jos am pus 2 poze cu procesorul, memoria și sistemul de operare al sistemului de calcul pe care am rulat testele:

```
student@pc:~$ lshw -short
WARNING: you should run this program as super-user.
H/W path           Device      Class      Description
=====================================================
                               system     Computer
/0                             bus        Motherboard
/0/0                           memory     4GiB System memory
/0/1                           processor  Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz
/0/2                           processor  Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz
/0/100                         bridge     440BX/ZX/DX - 82443BX/ZX/DX Host bridge
/0/100/1                       bridge     440BX/ZX/DX - 82443BX/ZX/DX AGP bridge
/0/100/7                       bridge     82371AB/EB/MB PIIX4 ISA
/0/100/7.1                     storage    82371AB/EB/MB PIIX4 IDE
/0/100/7.3                     bridge     82371AB/EB/MB PIIX4 ACPI
/0/100/7.7                     generic    Virtual Machine Communication Interface
/0/100/f                       display    SVGA II Adapter
/0/100/10          scsi32      storage    53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI
/0/100/11                      bridge     PCI bridge
/0/100/11/0                    bus        USB1.1 UHCI Controller
/0/100/11/1        ens33       network    82545EM Gigabit Ethernet Controller (Copper)
/0/100/11/3                    bus        USB2 EHCI Controller
/0/100/11/5        scsi3       storage    SATA AHCI controller
/0/100/11/5/0.0.0  /dev/cdrom  disk       VMware SATA CD01
/0/100/15                      bridge     PCI Express Root Port
```

Fig. 1. Procesorul si Memoria

```
student@pc:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="20.04.1 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.1 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
```

Fig. 2. Sistemul de Operare

3.3 Ilustrarea rezultatelor evaluării soluțiilor pe setul de teste

O să notez "Cea mai eficientă metodă pentru grafurile orientate aciclice" cu MEAInDAG (Most Efficient Algorithm In Directed Acyclic Graphs).
Am ales să nu reprezint grafic rezultatele primelor 7 teste deoarece acestea au fost concepute pentru a testa corectitudinea algoritmilor, nu eficiența (având un număr de noduri cuprins între 5 și 9, timpii de execuție sunt aproape identici pentru toți algoritmii)

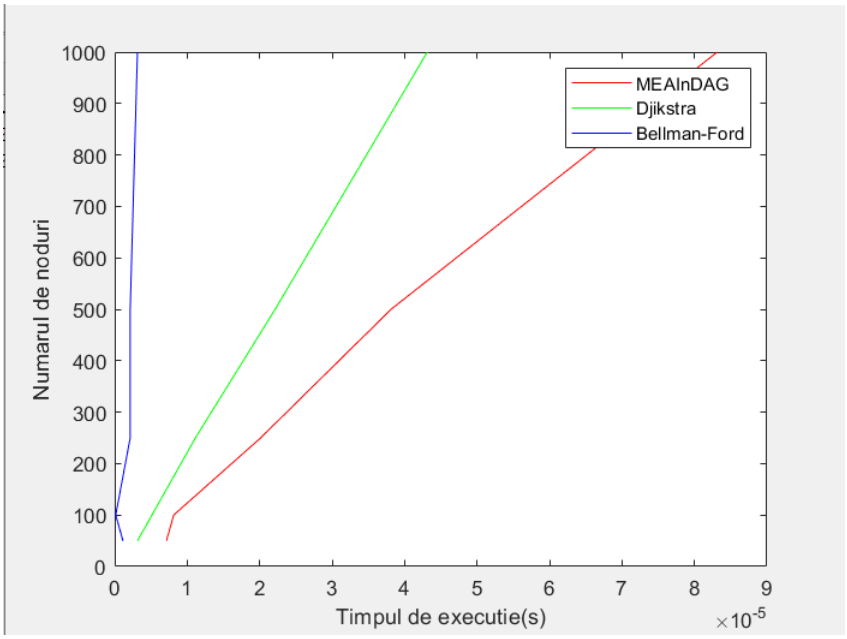
Nr.test	Djikstra	Bellman-Ford	MEAInDAG
1.	0.00000214	0.00000214	0.00000214
2.	0.00000214	0.00000114	0.00000214
3.	0.00000214	0.00000114	0.00000214
4.	0.00000214	0.00000114	0.00000114
5.	0.00000214	0.00000214	0.00000214
6.	0.00000114	0.00000114	0.00000114
7.	0.00000114	0.00000114	0.00000114

Tabelul timpilor de executie pentru testele 1-7 în secunde

– Pentru testele 8 - 12 (grafuri cu 0 muchii):

Nr.noduri	Djikstra	Bellman-Ford	MEAInDAG
50	0.00000314	0.00000114	0.00000714
100	0.00000514	0.00000014	0.00000814
250	0.00001114	0.00000214	0.00002014
500	0.00002214	0.00000214	0.00003814
1000	0.00004314	0.00000314	0.00008314

Tabelul timpilor de executie pentru testele 8-12 în secunde

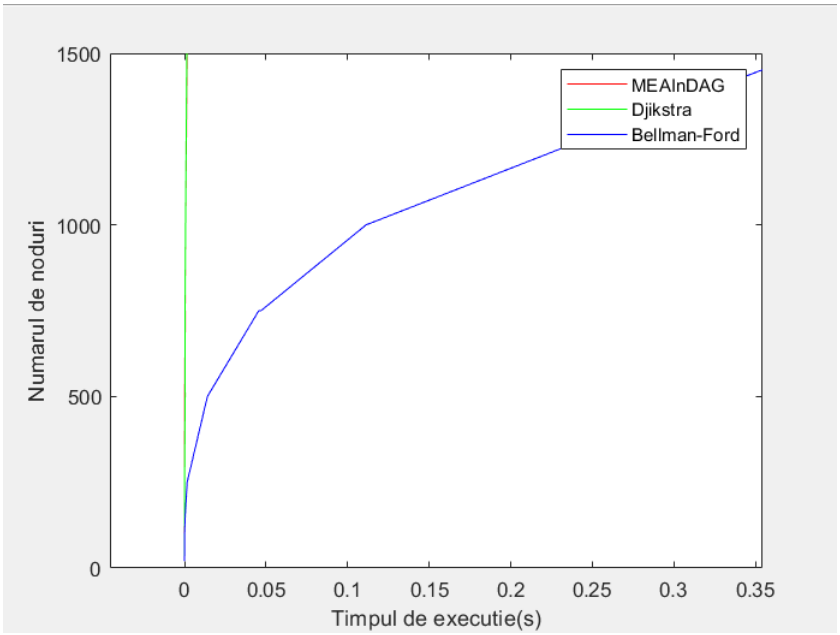


Graficul testelor 8-12 (grafuri cu 0 muchii) în funcție de timpul de rulare și numărul de noduri

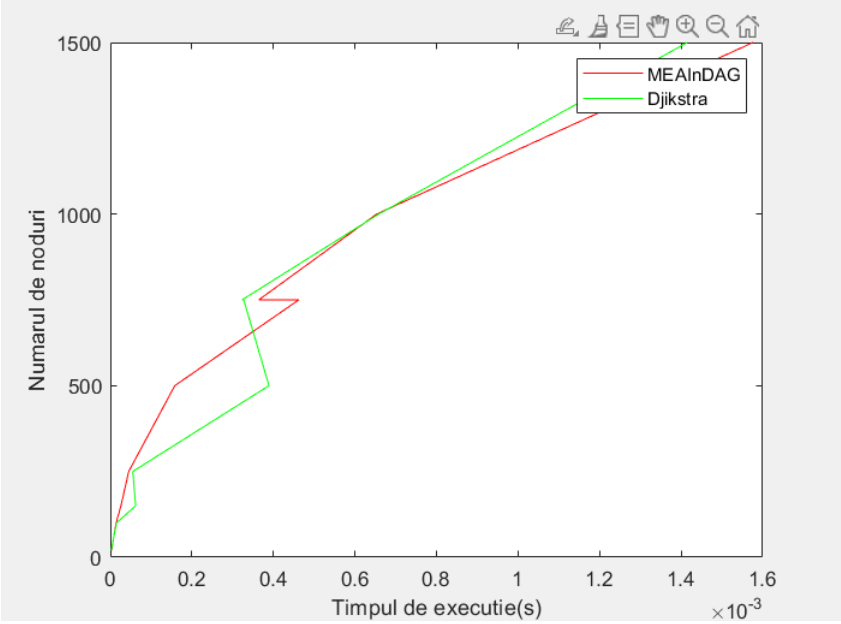
– Pentru testele 13 - 21 (grafuri rare):

Nr.noduri	Djikstra	Bellman-Ford	MEAInDAG
20	0.00000214	0.00000114	0.00000314
100	0.00001514	0.00009414	0.00001414
150	0.00006214	0.00040914	0.00002614
250	0.00005514	0.00162914	0.00004514
500	0.00038914	0.01404314	0.00015814
750	0.00032714	0.04553514	0.00046214
750	0.00032314	0.04690814	0.00036414
1000	0.00065814	0.11130614	0.00065314
1500	0.00141514	0.38000714	0.00157614

Tabelul timpilor de executie pentru testele 13-21 în secunde



Graficul testelor 13-21 (grafuri rare) în funcție de timpul de rulare și numărul de noduri

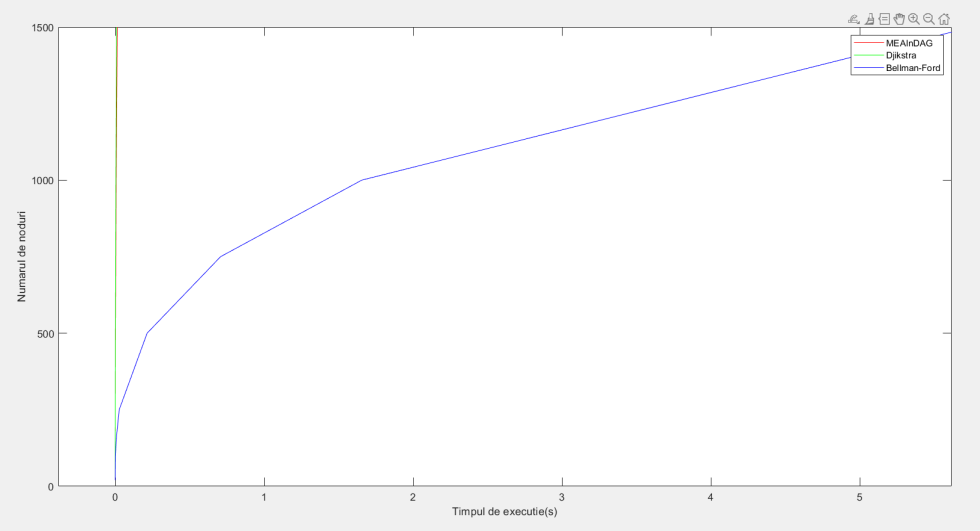


Diferența dintre MEAInDAG și Djikstra pentru grafuri rare

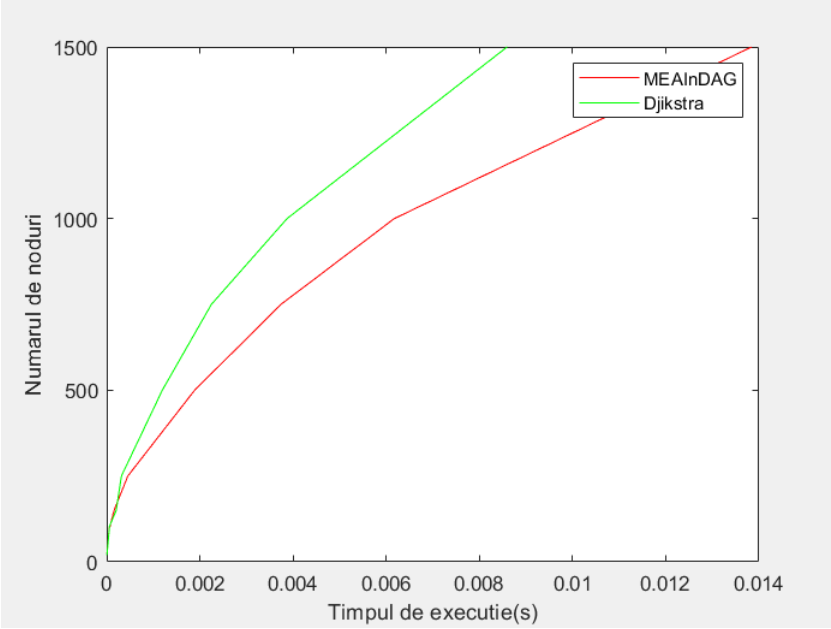
– Pentru testele 22 - 30 (grafuri dense):

Nr.noduri	Djikstra	Bellman-Ford	MEAInDAG
20	0.00000514	0.00007014	0.00000414
100	0.00005814	0.00164014	0.00005514
100	0.00005014	0.00161414	0.00006514
150	0.00020414	0.00589514	0.00015814
250	0.00031614	0.02634014	0.00045414
500	0.00119414	0.21388114	0.00188814
750	0.00224714	0.70729714	0.00374114
1000	0.00387214	1.65544114	0.00617414
1500	0.00859514	5.75186514	0.01384514

Tabelul timpilor de executie pentru testele 22-30 în secunde



Graficul testelor 22-30 (grafuri dense) în funcție de timpul de rulare și numărul de noduri

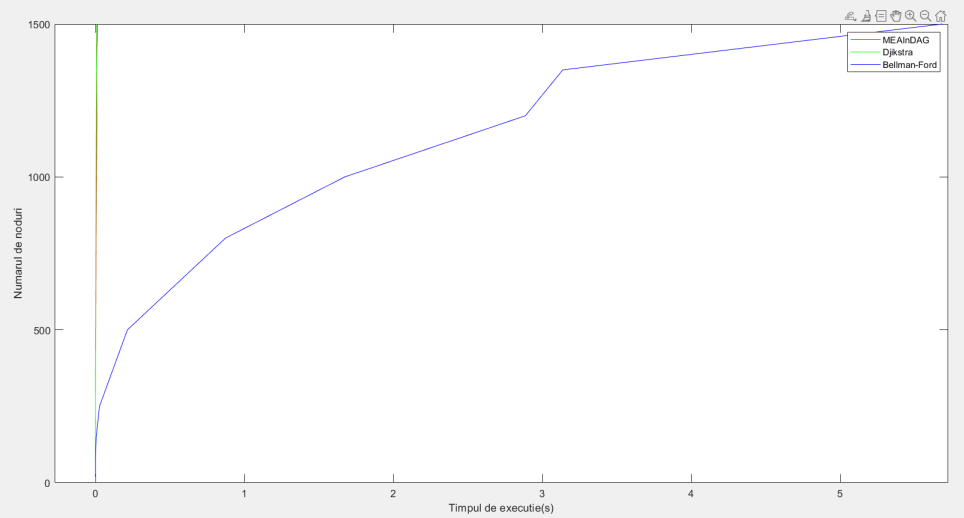


Diferența dintre MEAInDAG și Djikstra pentru grafuri dense

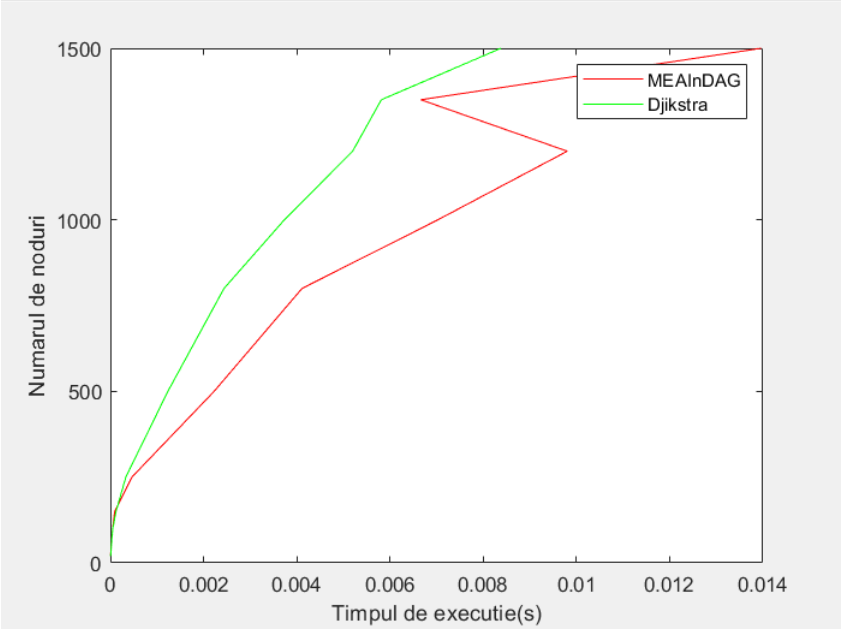
– Pentru testele 31 - 41 (grafuri random):

Nr.noduri	Djikstra	Bellman-Ford	MEAInDAG
20	0.00000614	0.00001314	0.00000414
70	0.00003214	0.00055014	0.00003014
100	0.00005214	0.00152014	0.00005014
150	0.00012214	0.00490414	0.00009314
250	0.00033214	0.02702614	0.00046214
500	0.00123414	0.21445014	0.00222614
800	0.00243914	0.87298814	0.00411314
1000	0.00373314	1.67333514	0.00702614
1200	0.00519814	2.88646114	0.00980314
1350	0.00581414	3.13782014	0.00666114
1500	0.00837914	5.68495914	0.01396314

Tabelul timpilor de executie pentru testele 31-41 în secunde



Graficul testelor 31-41 (grafuri random) în funcție de timpul de rulare și numărul de noduri



Diferența dintre MEAInDAG și Djikstra pentru grafuri random

3.4 Prezentarea succintă a valorilor obținute

În cazul grafurilor cu 0 muchii, se observă faptul că Bellman-Ford este mai eficient decât ceilalți algoritmi deoarece are o complexitate de $O(|V|)$. Acest lucru se datorează faptului că bucla în care se parcurg toate muchiile grafului nu se

execută niciodată deoarece nu există muchii. Se vor parcurge doar cele V noduri, de unde și complexitatea. Deși nici în cazul algoritmilor Djikstra si MEAInDAG nu se parcurg muchiile, complexitatea lor este mai mare deoarece Djikstra reactualizează coada de priorități după parcurgerea fiecărui nod, iar MEAInDAG sortează topologic nodurile.

În cazul grafurilor rare, timpul de execuție al algortimului Bellman-Ford este mult mai mare decât în cazul celorlalți 2 algoritmi, care au timpi de execuție destul de apropiați.

Observație! Faptul că, pentru un graf cu mai multe noduri, timpii de execuție ai algoritmilor Djikstra și MEAInDAG sunt mai mici decât în cazul unui graf cu mai puține noduri (apare acel "S" în grafic), este datorat funcției rand(), care contribuie la generarea de grafuri cu un "grad de raritate" diferit (graful cu un număr mai mare de noduri este mai rar decât graful cu un număr mai mic de noduri). În orice caz, începând cu un număr mai mare de noduri (1000 în cazul nostru), graficele timpilor de execuție ai celor 2 algoritmi se stabilizează.

Graficele testelor pentru grafuri dense și grafuri random sunt destul de asemănătoare cu graficul testelor pentru grafuri rare, singura diferență fiind că, în cazul grafurilor dense, graficele timpilor de execuție ai algoritmilor Djikstra și MEAInDAG nu mai au puncte de inflexiune(acel "S" în grafic). Acest lucru este datorat faptului că grafurile generate nu mai depind de funcția rand() (elementele de deasupra diagonalei principale a matricei de adiacență sunt diferite de 0. În cazul grafurilor rare și în cazul grafurilor random, matricea de adiacență are și elemente nule deasupra diagonalei principale).

Cu toate că, teoretic, MEAInDAG este cel mai eficient algoritm de determinare a distanței minime de la un nod la toate celelalte noduri dintr-un graf orientat aciclic, graficele nu spun același lucru (pentru un număr de noduri mai mare de 750 în cazul grafurilor rare, 250 în cazul grafurilor dense și random), sugerând faptul că Djikstra ar fi mai rapid. Acest lucru este datorat faptului că MEAInDAG folosește sortare topologică pentru a putea parcurge nodurile și muchiile grafului în $O(|V| + |E|)$. Sortarea topologică este un proces destul de costisitor din punctul de vedere al timpului execuției, având complexitate $O(|V|+|E|)$. Astfel, timpul de execuție al algoritmului este dublat.

4 Concluzii

Bellman-Ford este cel mai ușor de implementat dintre toți algoritmii aleși și poate gestiona și muchii de cost atât pozitiv, cât și negativ. Această versatilitate crescută îi oferă algoritmului potențialul reprezentării unui număr mai mare de situații din viața reală decât Djikstra. Cu toate acestea, algoritmul Bellman-Ford este mult mai lent decât Djikstra sau Cea mai eficientă metodă pentru grafuri orientate aciclice, iar aplicabilitatea acestuia în situațiile din viața reală,

la ora actuală, nu este mai mare decât a algoritmului Djikstra (sunt destul de rare grafurile cu muchii de cost negativ [13]). Algoritmul "Cea mai eficientă metodă pentru grafuri orientate aciclice", deși este cel mai rapid dintre toți algoritmii, nu are o aplicabilitate foarte mare, fiind util doar când vrem să facem anumite lucruri într-o anumită ordine (când există relații de dependență între lucrurile pe care vrem să le facem [9])

Acestea fiind zise, înainte de alegerea unui algoritm aș analiza problema pe care trebuie să o rezolv. Dacă problema presupune rezolvarea anumitor lucruri într-o anumită ordine, atunci aș alege Cea mai eficientă metodă pentru grafuri orientate aciclice. Altfel, aș încerca să rezolv problema folosind algoritmul Djikstra. Dacă problema nu poate fi rezolvată folosind algoritmul Djikstra, aș încerca să o rezolv folosind algoritmul Bellman-Ford.

References

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms
2. https://andrei.clubcisco.ro/2pa/laboratoare/Laborator%208_-_drumuri_minime_1.pdf
3. https://ocw.cs.pub.ro/courses/pa/laboratoare/laborator-09
4. https://www.techiedelight.com/single-source-shortest-paths-dijkstras-algorithm/
5. https://www.easycalculation.com/operations-research/shortest-path-calculator.php
6. https://www.geeksforgeeks.org/shortest-path-for-directed-acyclic-graphs/
7. https://www.geeksforgeeks.org/dijkstras-algorithm-for-adjacency-list-representation-greedy-algo-8/?ref=lbp
8. https://en.wikipedia.org/wiki/Directed_acyclic_graph
9. https://www.quora.com/Why-do-we-need-to-find-topological-sort-to-find-the-shortest-path-in-DAG-Wha
10. https://www.baeldung.com/cs/dijkstra-time-complexity
11. https://iq.opengenus.org/shortest-path-using-topological-sort/
12. https://www.baeldung.com/cs/dijkstra-vs-bellman-ford
13. https://medium.com/@nico_72892/an-analysis-of-dijkstras-algorithm-fcd64fc827c7