Vilniaus Universitetas
Informatikos institutas

Benediktas Belevičius
informatikos specialybė,
II kursas, II grupė

Ausų algoritmas

Vilnius – 2023

Contents

Įvadas

Šiame darbe tiriama Jens M. Schmidt'o algoritmo, skirto grafo jungumui įvertinti, veikimas, efektyvumas ir panaudojimas.

Tyrimo tikslas yra suprasti ir bandymais įvertinti empyrini algoritmo sudėtingumą laiko prasme bei ištirti, kokį poveikį skirtingi kriterijai turi grafų išskirstymui ausimis. Tyrimui naudojau tik paprastuosius, šių rūšių grafus:

1. Jungiuosius grafus
2. Nejungiuosius grafus
3. Pilnus grafus

Tyrimą sudaro šie etapai:

1. Algoritmo pseudokodo empyrinė analizė
2. Algoritmo efektyvumo analizė pasitelkiant bandymus

Tyrimo metu naudota (*python*) programavimo kalba parašytas programinis kodas.

1 Apibrėžimai

Norint tęsti reikia apibrėžti kelias sąvokas. Visų pirma, kas gi yra ausis grafų kontekste

Apibrėžimas 1 (Ausis) *Kelias* $P_n \in G(V, E)$*, kurio galų viršūnės* $x, y \in H$*, kur* $H \subset G$*.*
Pastaba: *x ir y gali būti ta pati viršūnė, todėl ciklas irgi yra ausis.*

Mano naudojamas algoritmas grąžina blokų sąrašą, todėl reikia suprasti Bloko reikšmę

Apibrėžimas 2 (Blokas) *Jungusis grafas, turintis bent dvi viršūnes ir neturintis iškarpos taškų. Pvz:* K_2*,* C_3 *ir t.t. Blokai neturi tiltų (išskyrus* K_2*).*

Algoritmo tikslas patikrinti ar egzistuoja išskaidymas ausimis, todėl reikia suprasti, kas tai yra

Apibrėžimas 3 (Išskaidymas ausimis) *Grafo užrašymo būdas kaip sąrašo, kur kiekvienas narys yra blokas, o kiekvienas blokas yra sąrašas ausų, kur pirma ausis yra ciklas, o toliau seka ausys kurios galų viršūnes priklauso ciklui arba pirmiau sąraše esančioms ausims.*

Sąrašo pavyzdys:

$$[[C_3, P_4, P_2], [C_8, P_8, P_2, P_2]]$$

Šį grafą sudaro du blokai:
Pirmasis blokas yra sudarytas iš ciklo C_3 *ir ausų* P_4 *ir* P_2
Antrasis blokas yra sudarytas iš ciklo C_8 *ir ausų* P_8*,* P_2 *ir* P_2
Pastaba: *Blokų sąraše nematysime tikslių* K_2 *blokų, nes šie neturi ciklo, vietoje jų rasime tuščius blokus žymimus kaip* [ ]

2 Algoritmai

Schmidt'o ausų radimo algoritmas naudoja DFS (paiešką gylyn), todėl reikia apsirašyti DFS radimo algoritmą:

Algorithm 1 DFS(G, root)

Duota: Paprastasis grafas $G = (V, E)$, viršūnė
Rasti: DFS medį ir DFS medžio išraišką masyvu

1: $dfsArray := []$
2: $dfsArray \leftarrow root$
3: $dfsTree := Graph()$
4: $DFSRecursive(G, root, dfsArray, dfsTree)$

Pirmoji DFS funkcija kviečia šią, rekursyvią DFS radimo funkcijos dalį.

Algorithm 2 DFSRecrusive(G, root, dfsArray, dfsTree)

Duota: Paprastasis grafas $G = (V, E)$, viršūnė, dalinis DFS medis ir dalinis DFS masyvas
Rasti: DFS medį ir DFS medžio išraišką masyvu

1: $dfsArray := []$
2: $dfsArray \leftarrow root$
3: $dfsTree := Graph()$
4: **for all** $u \in ADJ(v)$ **do**
5: **if** $u \notin dfsArray$ **then**
6: $dfsArray \leftarrow u$
7: $dfsTree.edges() \leftarrow (u, root)$
8: $DFSRecursive(G, root, dfsArray, dfsTree)$
9: **end if**
10: **end for**
11: **return** CCB

Turėdami DFS algoritmą galime aprašyti Schmidt'o išskaidymo ausimis radimo algoritmą. Aprašysiu originalią ir paredaguotas šio algoritmo versijas. Originaliosios tikslas - patikrinti ar grafas yra 2-jungus. Jei grafas yra 2-jungus tuomet kiekviena jo viršūnė priklauso ciklui, o išskaidymas ausimis realiai tik nusako, kiekvieną viršūnę, kuri priklauso ciklui. Jeigu visos viršūnės priklauso, kuriai nors ausiai, tai grafas 2-jungus.

Algorithm 3 is2Connective(G, root)

Duota: Paprastasis grafas $G = (V, E)$, pradinę viršūnę
Rasti: Ar grafas 2-jungus

1: $visited := [root]$
2: $visitedEdges := []$
3: $dfsTree, dfsArray := DFS(G, root)$
4: $block := []$
5: **for all** $node$ **in** $dfsArray$ **do**
6: **for all** v **in** $ADJ(root)$ **do**
7: **if** $\not dfsTree.isBackedge((v, node) OR (v, node) in visitedEdges$ **then**
8: **continue**
9: **end if**
10: $visitedEdges \leftarrow (v, node)$
11: $ear := [node]$
12: $x := v$
13: **while true do**
14: $ear \leftarrow x$
15: **if** $x \in visited$ **then**
16: **break**
17: **end if**
18: $visited \leftarrow x$
19: $x := dfsTree.predecessors(x)[0]$
20: $visitedEdges \leftarrow (x, visited[-1])$
21: **end while**
22: $block \leftarrow ear$
23: **end for**
24: **end for**
25: **for all** $node \in G.nodes()$ **do**
26: **if** $node \notin block$ **then**
27: **return false**
28: **end if**
29: **end for**
30: **return true**

Mano paredaguotas algoritmas yra labai panašus. Skirtumas, tik kad vietoje pačio jungumo tikrinimo, algoritmas tiesiog suranda visus blokus ir bloką sudarančias ausis.

Algorithm 4 EarDecomposition(G, root)

Duota: Paprastasis grafas $G = (V, E)$, pradinę viršūnę
Rasti: Ar grafas 2-jungus

```
1: visited := []
2: visitedEdges := []
3: blocklist := []
4: for all root ∈ G.nodes() do
5:    visited ← root
6:    dfsTree, dfsArray := DFS(G, root)
7:    block := []
8:    for all node in dfsArray do
9:       for all v in ADJ(root) do
10:         if ¬dfsTree.isBackedge((v, node)) ∨ (v, node) ∈ visitedEdges then
11:            continue
12:         end if
13:         visitedEdges ← (v, node)
14:         ear := [node]
15:         x := v
16:         while true do
17:            ear ← x
18:            if x ∈ visited then
19:               break
20:            end if
21:            visited ← x
22:            x := dfsTree.predecessors(x)[0]
23:            visitedEdges ← (x, visited[−1])
24:         end while
25:         block ← ear
26:      end for
27:      blocklist ← block
28:   end for
29: end for
30: return blocklist
```

3 Algoritmų realizacija ir tyrimų eiga

Algoritmas įgyvendintas (*python*) programavimo kalba naudojantis NetworkX, Pyplot ir Pandas bibliotekomis. Grafai buvo generuojami NetworkX įgyvendintomis grafų generavimo funkcijomis, Pyplot buvo naudojama atvaizduoti gautiems duomenims, o Pandas naudota tvarkingam duomenų išdėstymui ir saugojimui failuose.

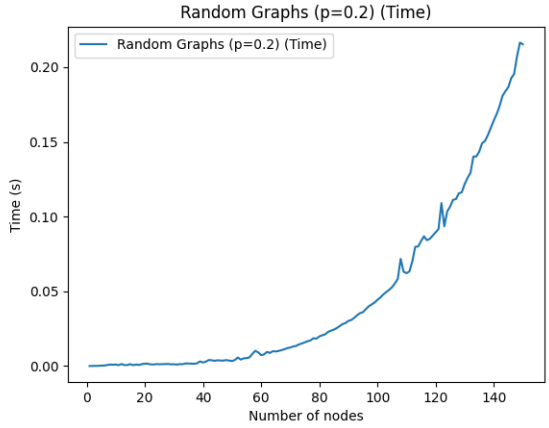
Tyrimai vykdyti pagal atitinkamus grafų kriterijus. Pastebėjau, kad algoritmo sudėtingumą nusakė viršūnių ir briaunų skaičius, todėl testuojant keičiau šių reikšmių vertes. Stebėjimo objektu (atsižvelgiant į pokyčius) tapo:

1. Vidutinis blokų skaičius
2. Vidutinis ausų kiekis bloke
3. Vidutinis ausies ilgis (viršūnėmis)
4. Didžiausio bloko dydis (ausimis)
5. Didžiausios ausies dydis (viršūnėmis)

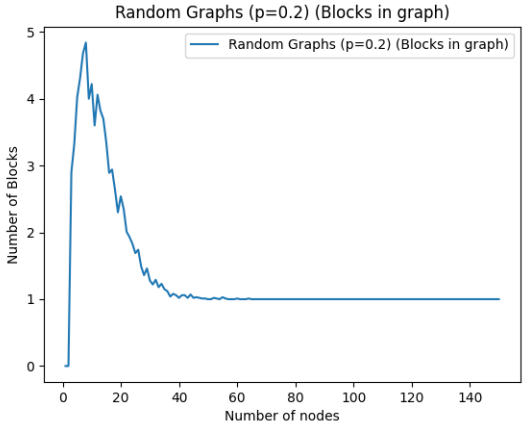
Pirmiausiai išnagrinėta buvo viršūnių pokytis, vėliau - briaunų tankis.
Programinį kodą galimą rasti: (`https://github.com/HussAreus/Ear-Decomposition-Analysis`)

4 Tyrimų rezultatai ir jų aptarimas

Nagrinėjant algoritmą galima pastebėti, kad kiekviena viršūnė ir kiekviena briauna yra aplankoma tik vieną kartą, todėl empyrinis algoritmo (laiko) sudėtingumas yra tik $\mathcal{O}(|V|+|E|)$. Žinoma, tuo reikia įsitikinti vykdant laiko efektyvumo analizę. Pradėjau nuo atsitiktinai sugeneruotų, jungiųjų grafų. Vykdant šią analizę, grafo dydį keičiau nuo |V| = 1 iki |V| = 100. Tikimybę, kad tarp viršūnių yra briauna, nustačiau kaip 0.2.



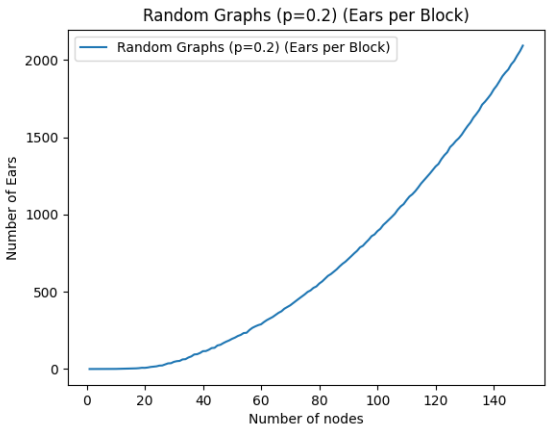
(a) Algoritmo vykdymo trukmė vienam testui



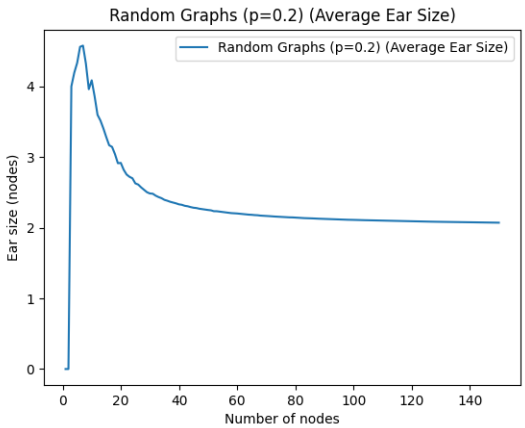
(b) Blokų skaičius grafe

Figure 1

Diagramoje (1a) matome kiek netikėtą vaizdą. Sudėtingumas laiko atžvilgiu yra panašesnis į $\mathcal{O}(n \log n)$ arba $\mathcal{O}(n^2)$. Taip nutinka dėl to, kad padidinus viršūnių skaičių, padidėja ir braiunų skaičius. Skaičiuojant, vidutiniškai $|E| = |V| \cdot \frac{|V|-1}{2} \cdot 0.2$, kas parodo kvadratinę |E| priklausomybę nuo |V|. Antrasis grafikas (1b) tik patvirtina, kad didėjant viršūnių skaičiui, didėja ir kaimynių skaičius, nes grafas tampa vis labiau jungus, kol galiausiai (daugumą bandymų) visos viršūnės priklauso vienam blokui.



(a) Vidutinis ausų skaičius bloke

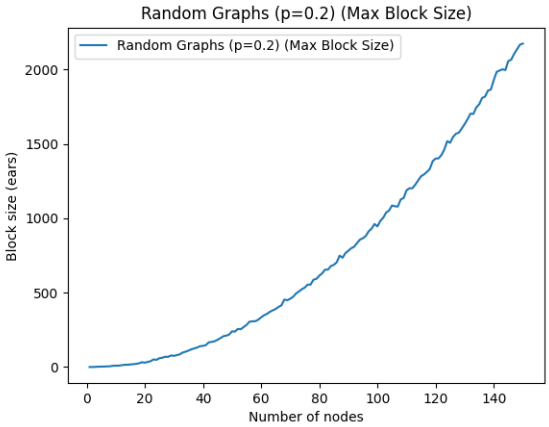


(b) Vidutinis ausiai priklausančių viršūnių skaičius

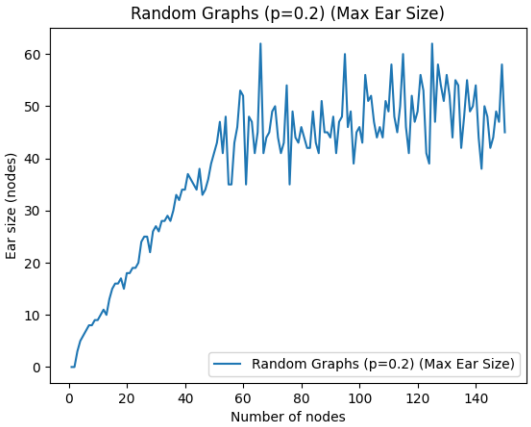
Figure 2

Šios dvi diagramos pavaizduoja kaip ausų dydis ir ausų kiekis kinta didinant grafą. Iš (1b) diagramos galime pastebėti, kad blokų skaičius mažėja, tuo tarpų ausų kiekis bloke didėja (žr 2a). Taip įvyksta dėl prieš tai paminėtos sąvybės. Didinant |V| skaičių didėja ir |E|, todėl vis didesnė tikimybė, kad DFS medyje rasti fundamentalūs ciklai į ausų sąrašą pridės mažo dydžio ausis, tačiau tų ausų bus daug. Tai patvirtina antra diagrama (2b). Matome, kad vidutinis ausies dydis sumažėja iki 2, kas yra mažiausia įmanoma ausis.

Pasuktinės dvi diagramos (3a, 3b) tik dar kartą parodo, kad didinant viršūnių skaičių, Bloke esančių ausų skaičius didėja, tuo tarpu didžiausiai ausiai tai didelės įtakos nedaro, nes dauguma ausų yra minimalaus, 2 viršūnių dydžio.



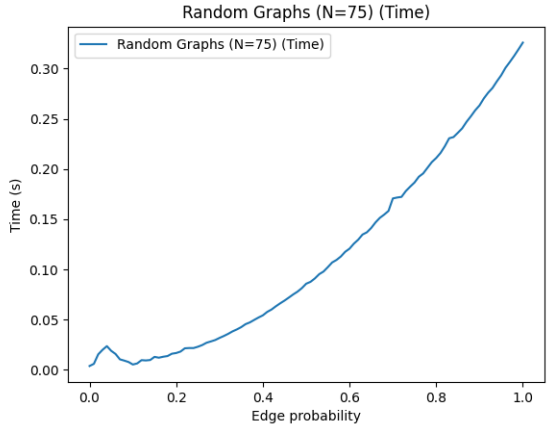
(a) Didžiausias blokas (ausimis)



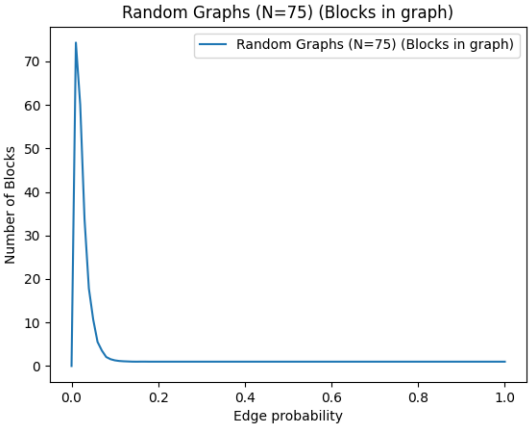
(b) Didžiausia ausis (Viršūnėmis)

Figure 3

Pastebėjęs, kad didžiausią įtaką skaičiavimams padarė briaunų skaičius, toliau tyriau kaip grafo viršūnių dažnis paveikia statistiką. Tyrimams pasirinkau 75 viršūnių dydžio grafą, o tikimybę, kad tarp dviejų viršūnių yra briauna, keičiau nuo 0 iki 1, kas 0.01. Atkreipsiu dėmesį, kad šį kartą grafai nėra būtinai jungieji.

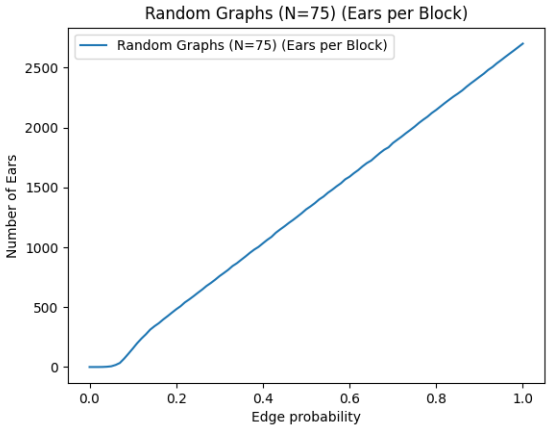


(a) Algoritmo vykdymo trukmė vienam testui

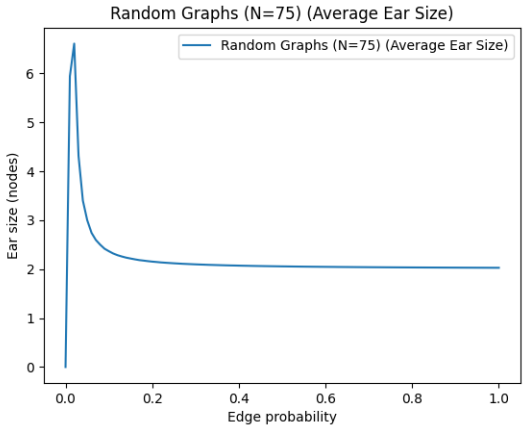


(b) Blokų skaičius grafe

Figure 4

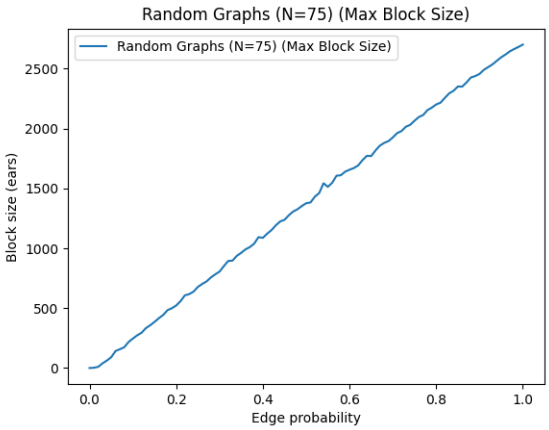


(a) Vidutinis ausų skaičius bloke

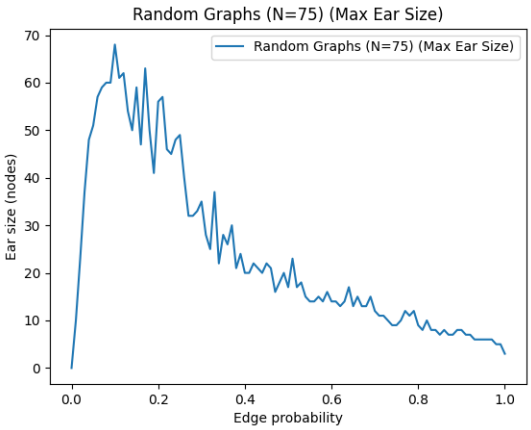


(b) Vidutinis ausiai priklausančių viršūnių skaičius

Figure 5



(a) Didžiausias blokas (ausimis)

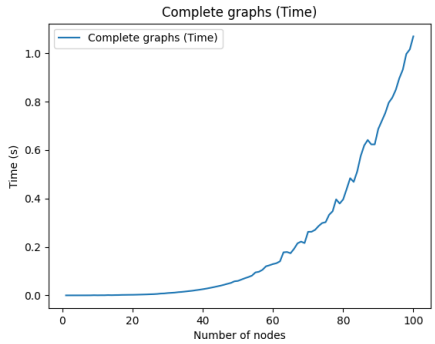


(b) Didžiausia ausis (Viršūnėmis)

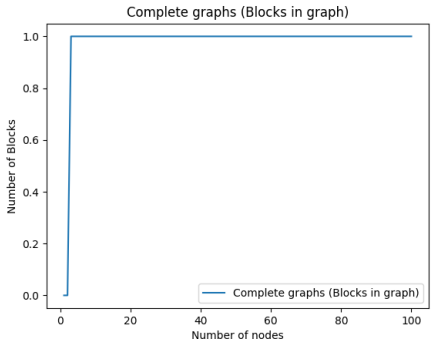
Figure 6

Iš šių diagramų pamatome, kad briaunų dažnis labiausiai paveikia Blokų skaičių (4b), Ausų skaičių bloke (5a) ir didžiausią ausį (6b). Blokų skaičius labai sparčiai sumažėja, kai grafas tampa jungesnis (savaime suprantama). Vidutinis ausų skaičius bloke tiesiškai didėja, dėl to, kad realiai tai tik padidiname 2 viršūnių dydžio ausų kiekį sulyg kiekviena nauja briauna. Ir. žinoma, įdomiausiai atrodantis Didžiausios ausies grafas parodo, kad kuo grafas panašesnis į pilną grafą, tuo mažesni DFS atrandami ciklai.

Pastebėjus prieš tai minėtą sąvybę, ištyriau ir pilnus grafus, Grafų dydžiai kito nuo 1 iki 100 viršūnių.

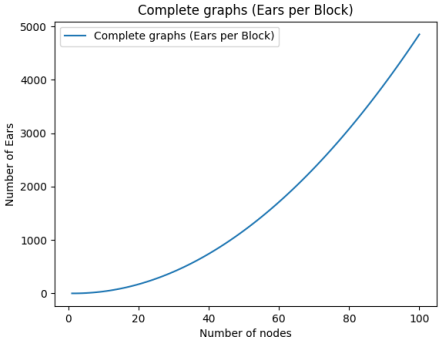


(a) Algoritmo vykdymo trukmė vienam testui

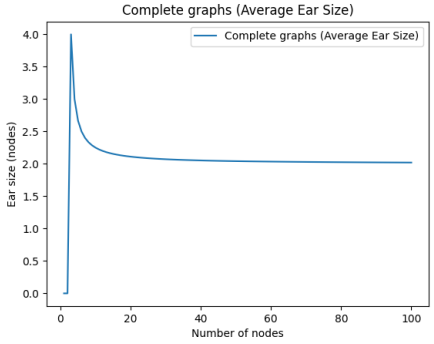


(b) Blokų skaičius grafe

Figure 7

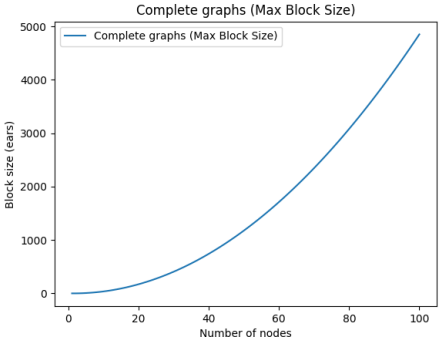


(a) Vidutinis ausų skaičius bloke

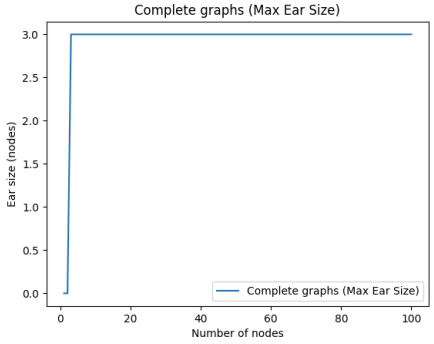


(b) Vidutinis ausiai priklausančių viršūnių skaičius

Figure 8



(a) Didžiausias blokas (ausimis)



(b) Didžiausia ausis (Viršūnėmis)

Figure 9

Tai, deja, neparodė ypatingų duomenų apart akivaizdžių: pilnas grafas yra blokas (7b), Visi DFS eiliškumu aptinkami ciklai bus iki 3 viršūnių dydžio, nes visos viršūnės yra tarpusavyje sujungtos (9b).

Atlikus duomenų analizę, galima padaryti daugiausia trivialias išvadas:

1. Algoritmas iš tikrųjų yra $\mathcal{O}(|V| + |E|)$ sudėtingumo laiko atžvilgiu
2. Didinant braiunų dažnį:
   (a) Vidutiniškas ausies dydis artėja link 2
   (b) Blokų kiekis grafe artėja link 1
   (c) Ženkliai didėja ausų kiekis bloke
3. Įtaką didžiausios ausies dydžiui turi tik DFS šaknis ir paieškos algoritmas

Išvados

Šiame darbe mes atlikome Schmidt'o Išskaidymo ausimis algoritmo laiko sudėtingumo analizę, atlikome skirtingų grafų išskaidymų ausimis tyrimus, stebint kokią įtaką daro briaunų tankis.

- Schmidt'o algoritmas yra optimalus norint patikrinti ar grafas yra 2-jungus arba siekiant surasti grafo išskaidymą ausimis
- Schmidt'o algoritmas sudėtingumas yra $\mathcal{O}(|V| + |E|)$, tai parodė pseudokodo analizė ir laiko efektyvumo testai
- Grafų briaunų dažnis turi labai didelę įtaką algoritmo veikas greičiui, nes aptinkamų ciklų skaičius didėja, kai didėja dažnis. Tai parodė algoritmo rezultatai.

Programinį kodą galimą rasti: (`https://github.com/HussAreus/Ear-Decomposition-Analysis`)

Šaltiniai

References

[1] Jens M. Schmidt *A Simple Test on 2-Vertex- and 2-Edge-Connectivity*, 2019. https://arxiv.org/ftp/arxiv/papers/1209/1209.0700.pdf

[2] Ramchandra *Ramchandra's blog: Ear decomposition tutorial*, 2019. https://codeforces.com/blog/entry/80932