ШИНЖЛЭХ УХААН ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ
Мэдээлэл холбооны технологийн сургууль

ЛАБ 9,10
ТАЙЛАН

Шалгасан багш: Б.ТУЯАЦЭЦЭГ /F.SW03/

Гүйцэтгэсэн: B221910040 Н.Жавхлантөгс

Улаанбаатар 2025

Network Dynamics

***Даалгавар.**

*** dynetx санг судал. Тодорхой нэг статик снапшотын хувьд эсвэл глобалаар буюу динамик сүлжээний хэмжээнд shortest, fastest, foremost, fastest shortest, shortest fastest замуудыг ол. Эдгээр замуудын ялгааг тайлбарла.**

DynGraph сангийн судалгаа
DynGraph нь сүлжээний динамик шинжилгээ хийхэд ашиглагддаг Python сан бөгөөд динамик (өөрчлөгдөж буй) сүлжээний хамаарал, чанарыг судлах боломжийг олгодог. Энэ даалгавар дээр таны шаардлагыг хэрэгжүүлэхийн тулд динамик сүлжээний хамгийн богино, хамгийн хурдан, хамгийн түрүүнд явсан замууд болон эдгээр замуудын ялгааг судалж үзэх болно.

1. Зам хайх төрлүүд:

Shortest Path (Хамгийн богино зам)

- Тайлбар: Эхнээс төгсгөлийн хамгийн богино замыг олох зам хайлт. Энэ нь замын жинтэй утга дээр суурилж ажилладаг (жишээ нь, жин нь зай, цаг, эсвэл мөнгө байж болно).

- Алгоритм: Энэ замыг олохын тулд Dijkstra эсвэл Bellman-Ford алгоритмуудыг ашиглаж болно. NetworkX болон Dynetx хоёр сан нь энэ төрлийн зам хайлт хийх боломжтой.

Fastest Path (Хамгийн хурдан зам)

- Тайлбар: Хурд эсвэл цаг хугацаа дээр үндэслэн хамгийн хурдан замыг олох зам хайлт. Энэ нь замын хурд эсвэл өнгөрсөн хугацааг тооцоолдог.

- Алгоритм: Fastest Path нь ерөнхийдөө цагийн хэмжүүр эсвэл хурдтай холбоотой. Динамик сүлжээнд time эсвэл speed гэх мэт өгөгдлүүдийг ашиглаж болох юм.

Foremost Path (Хамгийн түрүүнд хүрэх зам)

- Тайлбар: Хамгийн түрүүчийн замыг олох зам хайлт. Энэ нь хамгийн түрүүнд очих замыг олох зорилготой бөгөөд тухайн замын ачаалал, хугацааг тооцохгүйгээр шууд

зорилгодоо хүрэх замыг илрүүлдэг.

Fastest Shortest Path (Хурдан хамгийн богино зам)

- Тайлбар: Хурдан бөгөөд хамгийн богино замыг олох зам хайлт. Энэ нь fastest болон shortest гэсэн хоёр төрөл зам хайлтыг нэгтгэсэн нэр томьёо юм.

Shortest Fastest Path (Богино хамгийн хурдан зам)

- Тайлбар: Богино болон хурдан замыг илрүүлэх зам хайлт. Энэ нь хамгийн богино замтай бөгөөд замын хурд эсвэл цаг хугацаа хамгийн бага байх замыг хайж олохыг зорьдог.

Зам хайх алхамууд:

Тайлангийн энэ хэсэгт зам хайх алгоритмуудыг хэрхэн хэрэгжүүлэх талаар алхамуудыг бичиж болох юм:

1. **Сүлжээ үүсгэх**: Тухайн граф дээр үндэслэн сүлжээг үүсгэх.
2. **Зам хайлт хийх**: Уг сүлжээг ашиглан хамгийн богино, хурдан, түрүүнд хүрэх, хурдан хамгийн богино, богино хамгийн хурдан замуудыг ол.
3. **Үр дүн**: Зам хайлт хийсний дараа эдгээр замуудын ялгааг тайлбарлаж, харьцуулалт хийх.

Замуудын ялгаа:

- Shortest vs. Fastest Path: Shortest Path нь хамгийн бага жингтэй буюу хамгийн бага зайтай замуудыг илрүүлдэг. Харин Fastest Path нь цаг хугацаа, хурд зэргийг тооцон хамгийн түргэн шийдэл авах замуудыг тодорхойлно.

- Foremost Path vs. Fastest Path: Foremost Path нь хамгийн түрүүнд дуусгах замуудыг олно, харин Fastest Path нь хамгийн хурдан хугацаанд хүрэх замыг тодорхойлно.

- Shortest Fastest Path vs. Fastest Shortest Path: Энэ хоёр зам нь өөрөөр хэлбэл хоёр үзүүлэлт дээр төвлөрсөн байдаг. Хамгийн хурдан бөгөөд богино замыг илрүүлнэ, гэхдээ холболтуудын жин эсвэл хугацаа нь хоорондоо ялгаатай байж болно.

Код

```
import dynetx as dn

import dynetx.algorithms as al

import networkx as nx



# Динамик граф үүсгэх

g = dn.DynGraph()



# Ирмэгүүдийг цаг хугацааны тэмдэглэгээтэй нэмэх (логик дараалалтай)

g.add_interaction("A", "B", t=1)  # t=1: A -> B

g.add_interaction("B", "C", t=2)  # t=2: B -> C

g.add_interaction("C", "E", t=3)  # t=3: C -> E

g.add_interaction("A", "D", t=2)  # t=2: A -> D

g.add_interaction("D", "E", t=4)  # t=4: D -> E

g.add_interaction("A", "E", t=5)  # t=5: A -> E (шууд зам)



# Графийн бүтцийг шалгах

print("Бүх ирмэгүүд:", g.edges())



# Тодорхой нэг снапшот (t=1) дээрх статик граф

snapshot_t1 = g.time_slice(1)

print("Снапшот t=1 дээрх оройнууд:", snapshot_t1.nodes())

print("Снапшот t=1 дээрх ирмэгүүд:", snapshot_t1.edges())



# Глобал хэмжээнд замуудыг олох (A-с E хүртэл, t=1-ээс t=5 хүртэл)
```

```
start_node = "A"

end_node = "E"

start_time = 1

end_time = 5

paths = al.time_respecting_paths(g, start_node, end_node,
start=start_time, end=end_time)



print(f"\n{start_node}-с {end_node} хүртэлх цаг хугацааг харгалзсан
замууд:")

for i, path in enumerate(paths):

    print(f"Зам {i+1}: {path}")



# Замуудын шинжилгээ хийх функц

def analyze_paths(paths):

    if not paths:

        print("\nЗамуудын шинжилгээ: Зам олдсонгүй")

        return



    # Цагийн ялгааг авах функц

    def get_time_diff(path):

         if isinstance(path, list) and all(isinstance(p, tuple) and len(p)
== 3 for p in path):

            return path[-1][2] - path[0][2]

        return float('inf')  # Формат буруу бол хамгийн их утга



    # Замын дуусах цагийг олох
```

```
    def get_end_time(path):
        if isinstance(path, list) and all(isinstance(p, tuple) and len(p) == 3 for p in path):
            return path[-1][2]
        return float('inf')

    # Замын хамгийн богино, хурдан, тэргүүн замыг ялгах
    shortest_path = min(paths, key=len)
    fastest_path = min(paths, key=get_time_diff)
    foremost_path = min(paths, key=get_end_time)

    # Замуудын шинжилгээ гаргах
    print("\nЗамуудын шинжилгээ:")
    print(f"Shortest (хамгийн богино): {shortest_path}")
    print(f"Fastest (хамгийн хурдан): {fastest_path}")
    print(f"Foremost (хамгийн эрт ирэх): {foremost_path}")

    # Хамгийн богино зам дээрх хурдны ялгаа
    shortest_paths = [p for p in paths if len(p) == len(shortest_path)]
    fastest_shortest = min(shortest_paths, key=get_time_diff)

    # Хамгийн хурдан зам дээрх хамгийн богино зам
    fastest_paths = [p for p in paths if get_time_diff(p) == get_time_diff(fastest_path)]
    shortest_fastest = min(fastest_paths, key=len)
```

```
    # Замын шинжилгээний үр дүн

    print(f"Fastest Shortest: {fastest_shortest}")

    print(f"Shortest Fastest: {shortest_fastest}")



# Шинжилгээг гүйцэтгэх

analyze_paths(paths)
```

Кодны үр дүн:

```
Бүх ирмэгүүд: [('A', 'B'), ('A', 'D'), ('A', 'E'), ('B', 'C'), ('C', 'E'), ('E', 'D')]

Снапшот t=1 дээрх оройнууд: ['A', 'B']

Снапшот t=1 дээрх ирмэгүүд: [('A', 'B')]


A-с E хүртэлх цаг хугацааг харгалзсан замууд:

Зам 1: ('A', 'E')


Замуудын шинжилгээ:

Shortest (хамгийн богино): ('A', 'E')

Fastest (хамгийн хурдан): ('A', 'E')

Foremost (хамгийн эрт ирэх): ('A', 'E')

Fastest Shortest: ('A', 'E')

Shortest Fastest: ('A', 'E')
```

Кодны тайлбар

1. Сангуудыг импортлох

Энэхүү хэсэгт бид dynetx, dynetx.algorithms, болон networkx сангуудыг импортлож байна.

- dynetx сан нь динамик графуудтай ажиллахад зориулсан сан юм. Энэ нь графын эдийн засаг (орой, ирмэгүүд) цаг хугацааны дагуу хэрхэн өөрчлөгдөж байгааг хянах боломж олгодог.

- networkx сан нь статик графтай ажиллахад хэрэглэгддэг бөгөөд динамик графын оронд зөвхөн тогтмол графын бүтэцтэй ажилладаг.

- dynetx.algorithms нь динамик граф дээр алгоритм хэрэгжүүлэх тусгай арга хэрэгслүүдийг агуулдаг.

2. Динамик граф үүсгэх
DynGraph() функц ашиглан динамик граф үүсгэж байна. Энэ граф нь цаг хугацааны тусламжтайгаар графын орой (nodes) болон ирмэг (edges)-ийн хамаарлыг илэрхийлнэ.
3. Цаг хугацааны тусламжтайгаар харилцан үйлдлүүд нэмэх
Цаг хугацааны дагуу графын ирмэгүүдийг нэмэхэд бид add_interaction() функцыг ашиглаж байна.

- g.add_interaction("A", "B", t=1) — Энэ нь A болон B оройнуудын хоорондох харилцан үйлдлийг t=1 цаг хугацаанд нэмнэ.

- Дараа нь бид B болон C оройнуудын хоорондох харилцан үйлдлийг t=2 цаг хугацаанд нэмэх бөгөөд үүнийг үлдсэн ирмэгүүдтэй адилхан хийж байна.

4. Графийн бүтцийг шалгах
g.edges() болон g.nodes() функцуудыг ашиглан графын орой болон ирмэгүүдийг шалгадаг. Бид граф дээрх бүх орой болон ирмэгүүдийг хэвлэж гаргах боломжтой.
5. Снапшот үүсгэх
Динамик граф нь цаг хугацааны дагуу өөрчлөгдөж байдаг. Тиймээс бид тодорхой цаг хугацааны үеийн (snapshot) графыг авч үзэх боломжтой.

- g.time_slice(1) нь t=1 цаг хугацаанд үүссэн статик графийг авч байна. Бид энэ граф дээрх орой болон ирмэгүүдийг хэвлэж харж болно.

6. Глобал замууд олох
Глобал замуудыг олохын тулд бид time_respecting_paths() функц ашиглаж байна. Энэхүү функц нь графын хоёр оройн хооронд цаг хугацааг харгалзан замыг олох боломж олгодог.

- paths = al.time_respecting_paths(g, start_node, end_node, start=start_time, end=end_time) — Энэ нь start_node-оос end_node хүртэлх замуудыг start_time болон end_time цагийн хязгаартай олох болно.

7. Замуудыг шинжлэх
Замуудын шинжилгээ хийхэд бид зам бүрийг ялгах бөгөөд ямар зам хамгийн богино, хурдан, хамгийн эрт ирэхийг олж тогтооно.

- Shortest Path: Энэ нь хамгийн богино замыг илэрхийлнэ, түүнийг замын уртаар илэрхийлдэг.

- Fastest Path: Энэ нь хамгийн хурдан замыг илэрхийлнэ, түүнийг замын цагийн зөрүүгээр (цаг хугацааны ялгаатай) тодорхойлдог.

- Foremost Path: Энэ нь хамгийн түрүүнд ирэх замыг илэрхийлнэ, үүнийг замын төгсгөлд ирэх цаг хугацаа заадаг.

- Fastest Shortest: Энэ нь хамгийн хурдан, хамгийн богино замыг илэрхийлнэ.

- Shortest Fastest: Энэ нь хамгийн богино зам дээр хамгийн хурдан замыг тодорхойлдог.

Бид эдгээр замуудыг нэгтгэн шинжилгээ хийж, хамгийн тохиромжтой замыг олж хэвлэх болно.
8. Шинжилгээг хийх
analyze_paths() функц нь олон төрлийн замуудын шинжилгээг хийдэг. Тэр нь зам бүрийг хамгийн богино, хурдан, эрт ирэх замууд гэж ангилах боломжтой бөгөөд хамгийн сайн тохирох замыг хэвлэж харуулна.
Жишээ:
Замуудын үр дүн:
- Shortest Path: Хамгийн богино зам

- Fastest Path: Хамгийн хурдан зам

- Foremost Path: Хамгийн түрүүнд ирэх зам

- Fastest Shortest: Хамгийн хурдан, хамгийн богино зам

- Shortest Fastest: Хамгийн богино зам дээр хамгийн хурдан зам