

**МОНГОЛ УЛСЫН ИХ СУРГУУЛЬ
ХЭРЭГЛЭЭНИЙ ШИНЖЛЭХ УХААН, ИНЖЕНЕРЧЛЭЛИЙН СУРГУУЛЬ
МЭДЭЭЛЭЛ, КОМПЬЮТЕРИЙН УХААНЫ ТЭНХИМ**

Ганхуяг Эрхэмбаяр

**Нийгмийн сүлжээнээс бүлэг илрүүлэх хоорондын
төвийн арга**

**(Betweenness Centrality for community detection in social
networks)**

Програм хангамж (D061302)
Бакалаврын судалгааны ажил

Улаанбаатар

2022 он

**МОНГОЛ УЛСЫН ИХ СУРГУУЛЬ
ХЭРЭГЛЭЭНИЙ ШИНЖЛЭХ УХААН, ИНЖЕНЕРЧЛЭЛИЙН СУРГУУЛЬ
МЭДЭЭЛЭЛ, КОМПЬЮТЕРИЙН УХААНЫ ТЭНХИМ**

**Нийгмийн сүлжээнээс бүлэг илрүүлэх хоорондын төвийн
арга**

**(Betweenness Centrality for community detection in social
networks)**

Програм хангамж (D061302)
Бакалаврын судалгааны ажил

Удирдагч: _____ Др. П.Далайжаргал

Хамтран удирдагч: _____

Гүйцэтгэсэн: _____ Г.Эрхэмбаяр (18B1NUM0992)

Улаанбаатар

2022 он

Зохиогчийн баталгаа

Миний бие Ганхуяг Эрхэмбаяр ”Нийгмийн сүлжээнээс бүлэг илрүүлэх хоорондын төвийн арга” сэдэвтэй судалгааны ажлыг гүйцэтгэсэн болохыг зарлаж дараах зүйлсийг баталж байна:

- Ажил нь бүхэлдээ эсвэл ихэнхдээ Монгол Улсын Их Сургуулийн зэрэг горилохоор дэвшүүлсэн болно.
- Энэ ажлын аль нэг хэсгийг эсвэл бүхлээр нь ямар нэг их, дээд сургуулийн зэрэг горилохоор оруулж байгаагүй.
- Бусдын хийсэн ажлаас хуулбарлаагүй, ашигласан бол ишлэл, зүүлт хийсэн.
- Ажлыг би өөрөө (хамтарч) хийсэн ба миний хийсэн ажил, үзүүлсэн дэмжлэгийг дипломын ажилд тодорхой тусгасан.
- Ажилд тусалсан бүх эх сурвалжид талархаж байна.

Гарын үсэг: _____

Огноо: _____

ГАРЧИГ

УДИРТГАЛ	1
1. ОНОЛЫН ХЭСЭГ	2
1.1 Нийгмийн сүлжээн дэх бүлэг / Community in social networks	2
1.2 ВС хэмжигдэхүүн / Betweenness centrality	4
1.3 Betweenness дээр суурилсан бүлэг илрүүлэх алгоритм (Girvan Newman)	6
2. GIRVAN NEWMAN АЛГОРИТМЫН ХУРДНЫ САЙЖРУУЛАЛТ	9
2.1 Girvan Newman алгоритм буюу GN	9
2.2 Олон ирмэг устгах	10
2.3 Betweenness эрэмбийг ойролцоолох	11
3. ХЭРЭГЖҮҮЛЭЛТ, ШИНЖИЛГЭЭ	13
3.1 Ашигласан өгөгдлүүд	13
3.2 Yp дүн	14
ДҮГНЭЛТ	21
НОМ ЗҮЙ	21
ХАВСРАЛТ	23
A. КОДЫН ХЭРЭГЖҮҮЛЭЛТ	24
B. БУСАД ГРАФИК YP ДҮНГҮҮД	31

ЗУРГИЙН ЖАГСААЛТ

1.1	Сүлжээн дэх бүлэг	2
1.2	Карате клубийн нийгмийн сүлжээг дүрслэх граф ба 34 орой нь гишүүдийг илтгэх ба тэдгээрийн хооронд нийт 78 ирмэг буюу гишүүдийн харилцан холбоог илэрхийлнэ	3
1.3	Сүлжээн дэх бүлэг	4
1.4	Node betweenness	5
1.5	Edge betweenness	6
1.6	Жишээ: GN алгоритмын ажиллагаа	7
1.7	Босоо тэнхлэг нь зөв ангилсан оройн хэмжээ ба хэвтээ тэнхлэг нь бүлэг хоорондын ирмэгүүдийн тоог илтгэнэ. Дөрвөлжин нь уламжлалт шаталсан хуваах арга, тойрог нь Girvan Newman	8
2.1	Олон ирмэг устгах / Multiple edge removal	10
2.2	Ирмэгүүдийн ВС эрэмбийг ойролцоолох нь	11
3.1	Карате клубын сүлжээ	16
3.2	Карате клуб үр дүн	16
3.3	Каратэ клубын үр дүн (нэг удаа устгах ирмэгийн тоог 3 болгож багасгасны дараа.)	17
3.4	Хэсэг бүлэг делфины харилцаа хамаарлын сүлжээ	17
3.5	Делфины сүлжээний үр дүн	18
3.6	Facebook найзын сүлжээ	18
3.7	Facebook найзын сүлжээн дээрх алгоритмуудын үр дүн	19
3.8	Хиймлээр үүсгэсэн графын μ тогтмолоос хамаарч өөрчлөгдөх хугацааны биелэлт. ▼ - $N = 30000$, ▲ - $N = 15000$, ● - $N = 3000$, ■ - $N = 250$ ба $\epsilon = 0.3$, $\delta = 0.1$ байна.	20
3.9	μ тогтмол 0.05 болон 0.2 байхын ялгаа.[12]	20

B.1	Facebook page dataset	31
B.2	Facebook page dataset γρ δγн	31
B.3	LastFm dataset	32
B.4	Advogado dataset	32

ХҮСНЭГТИЙН ЖАГСААЛТ

2.1	Girvan Newman алгоритмийн ажиллах хугацаа [5].	9
3.1	Ашигласан өгөгдлүүдийн мэдээлэл ба snap болон networkrepository-с өгөгдлөө авсан болно.	14
3.2	Ажилласан хугацааны үр дүн. GNC нь олон ирмэг устгах мөн BC ойролцоолох аргыг нийлүүлсэн алгоритм (эпсилон = 0.3, дельта = 0.5 үед). GNM нь олон ирмэг устгах алгоритм. Эхний 3 граф дээр 2 бүлэгт хуваасан бол сүүлийн 3 граф дээр 10 бүлэгт хуваасан	15
3.3	Графууд дээр GNC алгоритмын үр дүн GN алгоритмтай хэдэн хувь таарч байгааг харуулж буй гүйцэтгэл. /2 бүлэг/	15

УДИРТГАЛ

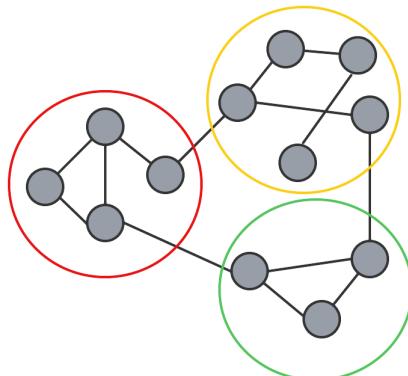
Орчин цагт нийгмийн сүлжээний хэрэгцээ, хэрэглээ огцом өсөж байгаа ба түүгээр зогсохгүй хоорондоо харилцан үйлчлэлд оршдог эд зүйлс обьектууд хүртэл сүлжээгээр илэрхийлэгдэж байгаа билээ. Нийгмийн сүлжээг судлах үед гарч ирдэг олон бодлого, асуудлын нэг бол нийгмийн сүлжээнээс бүлэг илрүүлэх юм. Бүлэг илрүүлэх гэсэн асуудлын хүрээнд олон төрлийн онцлог, шинж чанартай арга алгоритмууд байдаг ба энэхүү ажилд хоорондын аргыг ашиглан бүлэг илрүүлэх алгоритм дээр түлхүү ажилласан болно. Хоорондын аргаар бүлэг илрүүлэх алгоритмын хамгийн гол сул тал бол ажиллах хугацаа байдаг ба энэхүү ажиллах хугацааг хэрхэн оновчтой бууруулах тал дээр хийсэн судалгааны ажил болно. Уг алгоритмын ажиллах хугацааг багасгахын тулд олон ирмэг зэрэг устгах болон betweenness хэмжигдэхүүнийг ойролцоолох гэсэн талууд дээр ажилласан болно. Зорилго:

1. Сүлжээ, бүлэг илрүүлэх ойлголтуудыг судлах.
2. BC эрэмбэ, Girvan Newman алгоритмыг судлах
3. BC ойролцоолох алгоритмыг судалж, хэрэгжүүлэх.
4. Олон ирмэг устгах байдлаар GN алгоритмын хурдыг сайжруулах.
5. BC ойролцоолох алгоритмаа олон ирмэг устгах санаатай хослуулан хэрэглэх.
6. GNC буюу хослуулсан алгоритмаа зүгшрүүлэх, үнэлэх.

1. ОНОЛЫН ХЭСЭГ

1.1 Нийгмийн сүлжээн дэх бүлэг / Community in social networks

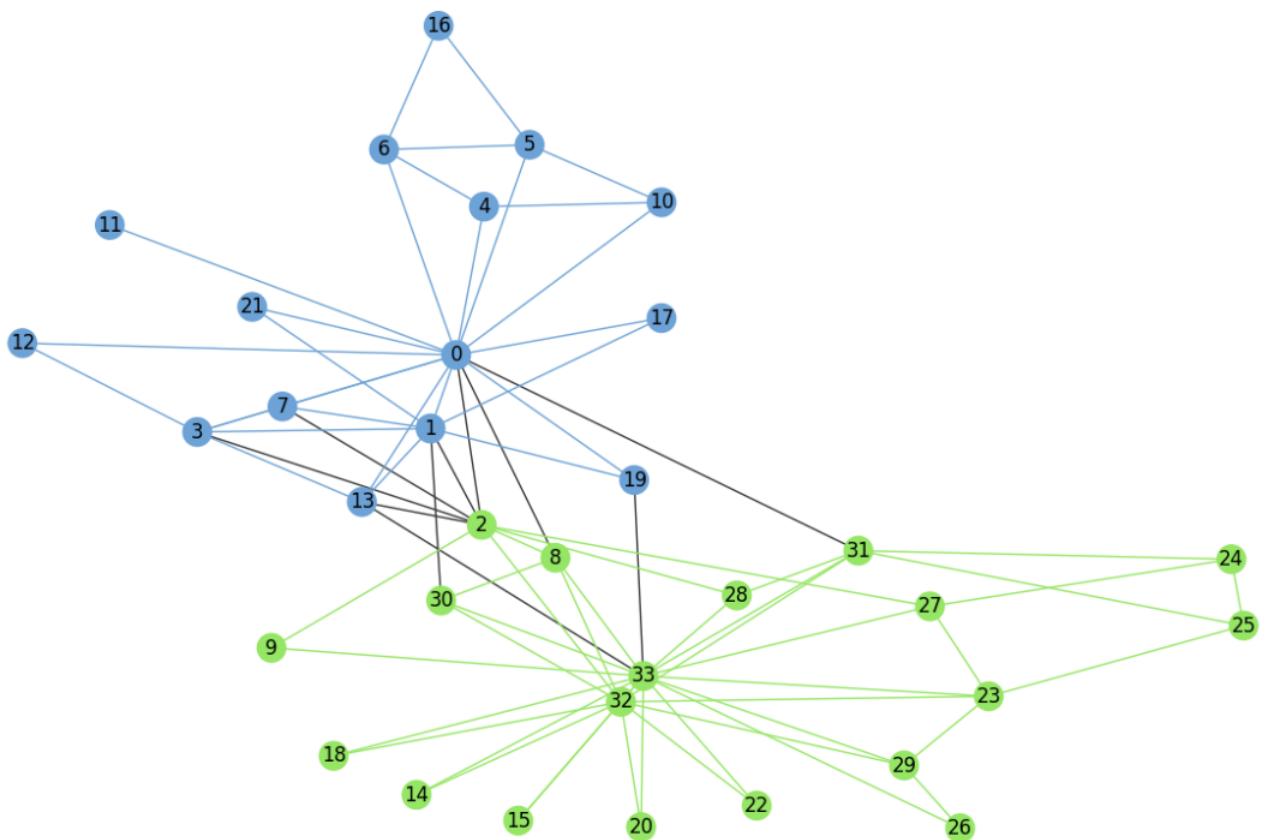
Олон судалгаанууд нийгмийн сүлжээний шинж чанарууд төвлөрдөг ба хамгийн нийтлэг шинж чанаруудаас дурдвал: small-world property, power-law degree distribution, network transitivity.[2] Энэхүү судалгааны ажил мөн эдгээр шинж чанаруудын нэг болох нийгмийн сүлжээний бүлэг гэсэн ойлголт дээр тулгуурлаж бичигдсэн болно. Сүлжээний бүлгийг судлах ажил нь олон салбарт гол асуудлуудын нэг байсаар ирсэн. Энэ асуудал нь нийгмийн сүлжээнд (интернет дээр байгаа харилцан хамаарлын судлал), биологийн судалгаа шинжилгээнд (уураг ба бодисын солилцооны дэд бүтэц) эсвэл технологийн асуудлууд (том дэд бүтцүүдийн сайжруулалт) г.м олон салбарт ач холбогдолтой судлагдахуун юм.[8] Жишээ нь найзын сүлжээг авч үзвэл хоорондоо найз болох хэсэг бүлэг хүмүүсийн цуглувлага болно. Эдгээр бүлгүүд нь хоорондоо хэд хэдэн хүмүүсээр холбогдож болох юм. Хоёр эсвэл түүнээс олон найзын хүрээлэлд хамарагддаг хүн байж болох учир. Энгийнээр хэлбэл, нийгмийн сүлжээн дэх бүлэг гэдэг маань сүлжээнд орших хэсэг бүлэг хоорондоо холбоотой эсвэл харилцан үйлчлэлд орсон объект, оройнуудын олонлог юм.



Зураг 1.1 Сүлжээн дэх бүлэг

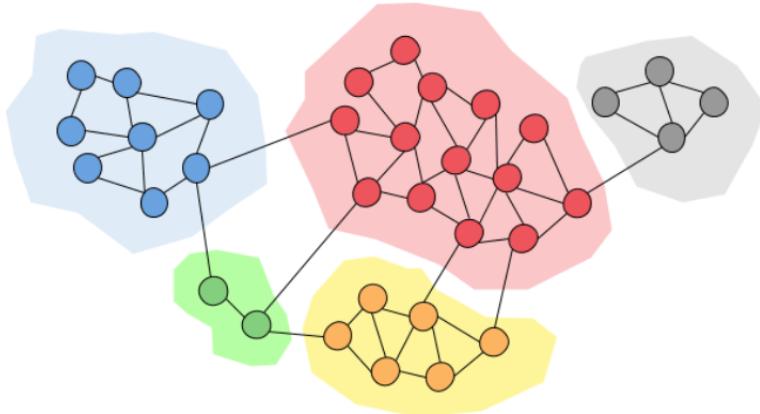
Бүлгүүд нь хоорондоо харилцаатай байгаа объект, оройнуудыг илрүүлэх боломжийг олгодог.

Жишээ нь: бүлэг нь сошиал сүлжээнд хамт нэг сургуульд явсан хүмүүсийг, өгүүлэл ишлэлийн сүлжээнд нэг өгүүллээс ишлэл татсан өгүүллүүд, уураг хоорондын харилцааны сүлжээнд орших модулиуд гэх мэт юуны сүлжээ байгаагаасаа хамаарч бүлэг нь үнэ цэнэтэй мэдээллийг агуулна. Бүлгийг илрүүлснээрээ харилцан үйлдлүүд нь адилхан объектуудыг илрүүлэх, байхгүй шинж чанарыг илрүүлэх, ирээдүйд бий болох холбоог таамаглах боломжийг олгох юм.[13] Сүлжээн дэх бүлэг алгоритмуудыг туршихад хэрэглэгддэг сонгодог жишээ болох "Zachary karate club" нийгмийн сүлжээг хоёр бүлэгт хуваасан байгааг жишээгээр харуулъя.



Зураг 1.2 Карате клубийн нийгмийн сүлжээг дүрслэх граф ба 34 орой нь гишүүдийг илтгэх ба тэдгээрийн хооронд нийт 78 ирмэг буюу гишүүдийн харилцан холбоог илэрхийлнэ.

Сүлжээнээс бүлэг илрүүлнэ гэдэг маань хэсэг бүлэг бүхий оройнуудыг ялгах явдал ба эдгээр оройнууд нь олон бүлэгт зэрэг харьялагдах мөн боломжтой юм. Учир нь бүлэгт харьялагдах оройнууд маань 2 төрлийн мэдээллээр ялгагдах боломжтой юм. Эхнийх нь: оройн талаарх мэдээлэл буюу шинж чанар нь юм. Харин нөгөө арга нь бол цэвэр сүлжээний бүтцэд анхаарах



[4]

Зураг 1.3 Сүлжээн дэх бүлэг

буюу оройнууд хэр нягт хоорондоо холбогдсон гэдгээс хамаарна. Шинж чанарын буюу эхний аргын хувьд “clustering” алгоритмууд яригдах ба объект хоорондын харилцан холбоог анхаардаггүй. Харин нөгөө талаас, бүлэг илрүүлэх арга, алгоритмууд нь сүлжээний бүтэц дээр анхаарч хоорондоо холбогдсон оройнууд дээр ажилладаг ба шинж чанарыг нь авч үздэггүй. Энэхүү судалгааны ажил нь сүлжээний бүтэц дээр суурилсан арга, алгоритмууд дээр төвлөрсөн болно.

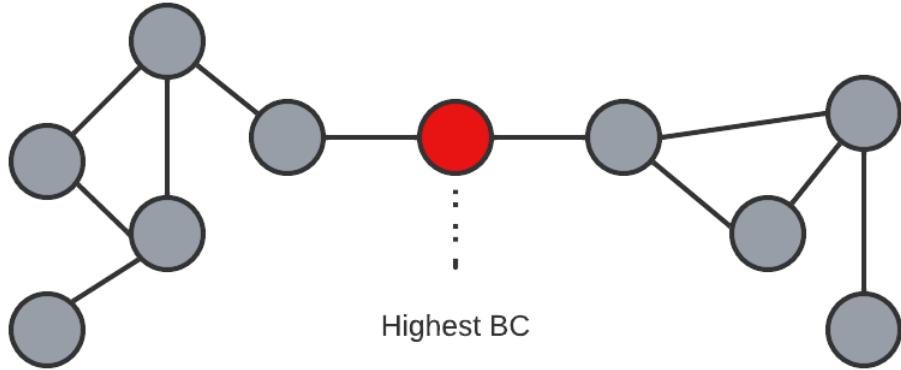
1.2 ВС хэмжигдэхүүн / Betweenness centrality

Сүлжээний анализын хамгийн нийтлэг бодлогуудын нэг бол тодорхой нэг оройн эсвэл ирмэгийн хэр чухал вэ гэдгийг нь тодорхойлох байдаг. Энэ хэмжигдэхүүнийг тодорхойлдог хамгийн нийтлэг аргууд бол closeness, stress, betweenness юм. Эдгээр дундаас хоорондын арга нь сүүлийн жилүүдэд social interaction сүлжээнүүд болон бусад төрлийн сүлжээнүүдэд ч их хэрэглэгдэж байна. Betweenness centrality гэх энэ хэмжигдэхүүнийг анх нягт биш графын шинж чанарыг тогтоохын тулд Freeman болон бусад хүмүүс оруулж ирсэн юм.[1] Betweenness нь аливаа нэгэн орой хэр хооронд вэ гэдэг асуултад хариулагддаг хамгийн дөт зам дээр суурилсан хэмжигдэхүүн юм. Өөрөөр хэлбэл эдгээр бүлгүүдийг холбож буй ирмэг эсвэл оройн ВС эрэмбэ хамгийн өндөр байна гэсэн үг юм. Боломжит бүх хос оройг авч үзэн тэдгээрийг холбосон хамгийн дөт замд орой маань хэдэн удаа орсон байгаагаар илэрхийлэгдэнэ.

Оройн betweenness ингэж тодорхойлогдоно [3]:

$$NB(v) = \sum_{v_i \in V} \sum_{v_j \in V/v_i} \frac{\sigma_{v_i v_j(v)}}{\sigma_{v_i v_j}}$$

Энд $\sigma_{v_i v_j(v)}$ нь v оройг дайрсан замын тоо ба $\sigma_{v_i v_j}$ нь нийт замын тоо юм. Ирмэгийн be-



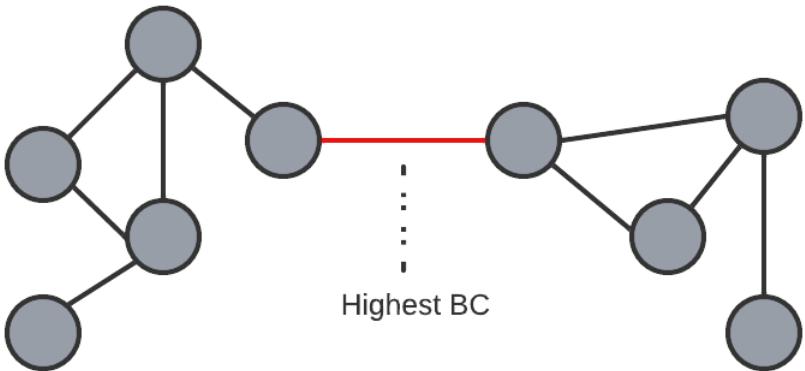
Зураг 1.4 Node betweenness

tweenness буюу ирмэгийн хэр хооронд байгааг тогтоохын тулд мөн адил граф дотор орших боломжит бүх хосыг авч үзэн тэдгээрийн дөт замд сонирхож байгаа ирмэг маань хэдэн удаа орсон гэдгээр илэрхийлэгдэнэ.

Ирмэгийн betweenness [3]:

$$EB(e) = \sum_{v_i \in V} \sum_{v_j \in V/v_i} \frac{\sigma_{v_i v_j(e)}}{\sigma_{v_i v_j}}$$

Энд $\sigma_{v_i v_j(e)}$ нь v ирмэгийг дайрсан замын тоо ба $\sigma_{v_i v_j}$ нь нийт замын тоо юм. Эндээс үзэхэд BC хэмжигдэхүүн нь өндөр байх тусам аливаа нэг орой эсвэл ирмэг нь сүлжээ, граф доторх бүлгүүдийг холбож өгч буй гүүр болж байна гэсэн үг. Жишээ нь хэсэг бүлэг хүмүүсийн найзын сүлжээг авч үзвэл өөрийн найзын бүлгээсээ гадна өөр найзын бүлгийн хүмүүстэй найз хүн байж болно. Эсвэл олон найзын бүлгийн хүмүүстэй найз хүн байж болно. Тэгвэл энэ хүн ба түүний найзын холбоо хамгийн өндөр BC зэрэгтэй буюу энэ хүн ба түүний холбоо нь сүлжээний хувьд бүлэг хоорондын орой эсвэл ирмэг(inter community edge or node) болох юм.

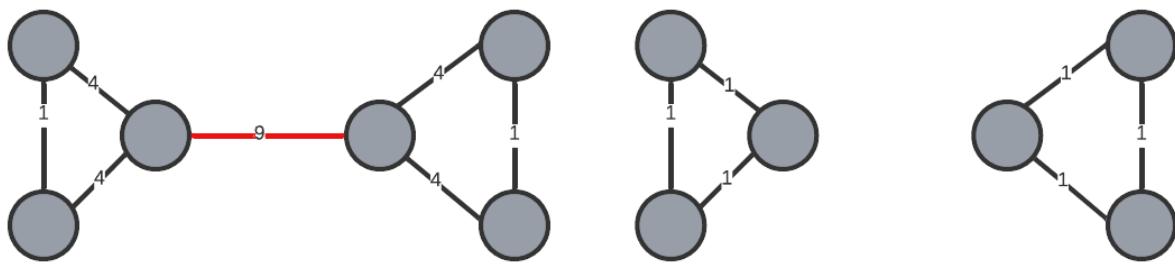


Зураг 1.5 Edge betweenness

1.3 Betweenness дээр сууринсан бүлэг илрүүлэх алгоритм (Girvan Newman)

Орчин цагт сүлжээ, графуудын бүлгийг илрүүлэхэд олон төрлийн онцлог, давуу талуудтай алгоритмууд ажиллаж байгаа билээ. Жишээ нь: fastgreedy, infomap, leading eigenvector, walktrap, spinglass, betweenness г.м. Мөн эдгээр алгоритмуудын нэг төлөөлөл бол ВС эрэмбийг ашиглан бүлэг илрүүлэх арга буюу Girvan Newman-н алгоритм юм[2]. Уг алгоритмыг Girvan тэргүүтэй хүмүүс санаачилсан ба уламжлалт бүлэг илрүүлэх аргуудтай харьцуулбал аливаа ирмэгийн хэр төв бэ (central) гэдэг асуултад биш харин эсрэгээрээ хамгийн төв биш ирмэгүүд дээр төвлөрч хамгийн хооронд ирмэгүүдийг олж илрүүлэхэд төвлөрнө. Ахнаасаа хоосон байсан оройн олонлогт хамгийн нөлөө ихтэй ирмэгүүдийг нэмэх байдлаар бүлгийг үүсгэхийн оронд анхны графаасаа ирмэгүүдийг устгах байдлаар бүлгүүдийг ялгах зарчмаар ажиллах юм. Оройн ВС хэмжигдэхүүн нь Freeman тэргүүтэй хүмүүс санаачилсан ба M. Girvan ба M. E. J. Newman нь энэхүү ВС хэмжигдэхүүнийг ирмэг дээр буулгаж өөрсдийн алгоритмдаа ашигласан юм. Хэрэв хоёр оройн хооронд нэгээс олон дөт зам оршин байвал бүх зам доторх ирмэгүүд ижил жинтэй гэж үзнэ. Хэрэв сүлжээ нь бүлгүүдийг агуулдаг мөн эдгээр бүлгүүд нь цөөхөн бүлэг хоорондын ирмэгүүдээр сул холбогдсон байвал бүх дөт замууд эдгээр ирмэгээр дамжих нь гарцаагүй. Тийм учир бүлгүүдийг холбож буй ирмэгүүд нь хамгийн өндөр ВС

эрэмбээтэй байна. Эдгээр ирмэгүүдийг устгаснаар бүлгүүдийг нэг нэгээс нь салгаж илрүүлэх юм. Энэ алгоритмын алхмууд нь сүлжээний бүх ирмэгийн betweenness эрэмбийг бодож олох ба betweenness өндөртэй ирмэг нь бүлэг хоорондыг холбосон inter community edge байх магадлал өндөр тул хамгийн өндөр betweenness-тэй ирмэгийг устган ахин шинээр бодож үүнийгээ ирмэг үлдэхгүй хүртэл давтах байдлаар ажиллана[2]. ВС эрэмбийг бodoход Newman-ы санаачилсан алгоритмыг[7] ашиглах ба энэ нь m ирмэгтэй n оройтой графын хувьд $O(mn)$ хугацаанд ажиллана. Ирмэг болгоныг устгах үед ВС тооцоолох алхам хийгдэх учир нийт хугацаа нь нийт ирмэгийн тоо дахин буюу $O(m^2n)$ болно. Гэхдээ давталт хийгдэх тутам нэг ирмэг устгаж дахин ВС-г тооцоолох үед зөвхөн байгаа ирмэгүүдийн эрэмбийг бодох тул энэ хамгийн удаан (worst case) биелэх хугацаанаас тодорхой хэмжээгээр бага байна.

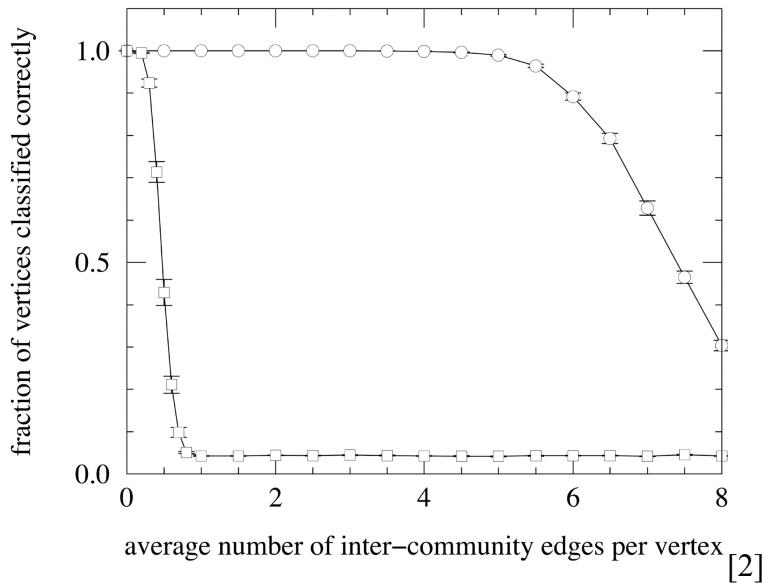


(a) GN ажиллахаас өмнө граф

(b) GN ажилласны дараа

Зураг 1.6 Жишээ: GN алгоритмын ажиллагаа

Girvan тэргүүтэй хүмүүс нь энэхүү санаачилсан алгоритмаа туршиж үзэхдээ хэд хэдэн жинхэнэ амьдрал дээрх нийгмийн сүлжээ мөн хиймлээр үүсгэсэн граф дээр туршиж үзсэн ба хиймлээр үүсгэсэн граф дээр уламжлалт hierarchical clustering арга болон өөрсдийн дэвшүүлсэн алгоритмын гүйцэтгэлийг харьцуулж байна.



[2]

Зураг 1.7 Босоо тэнхлэг нь зөв ангилсан оройн хэмжээ ба хэвтээ тэнхлэг нь бүлэг хоорондын ирмэгүүдийн тоог илтгэнэ. Дөрвөлжин нь уламжлалт шаталсан хуваах арга, тойрог нь Girvan Newman

Эндээс харахад GN алгоритм нь уламжлалт шаталсан хуваах аргатай харьцуулахад бүлэг хоорондын ирмэгийн хэмжээ нэмэгдэхэд чанараа илүү хадгалж буй харагдаж байна. Уг алгоритмын сул тал нь ВС эрэмбийг асар өндөр өртөгтэй тооцоолдог ба энэ нь том хэмжээний графууд дээр хэрэглэх боломжгүй болгож байгаа юм. Хэрэв энэхүү арга алгоритмын бүлэг хооронд орших ирмэгүүд дээр төвлөрөх гол санааг нь гээхгүйгээр хадгалж ВС эрэмбийг тооцоолох илүү хямд арга эсвэл нэг бодсон ВС эрэмбийг дахин ашиглах байдлаар ажиллах хугацааг нь багасгавал том хэмжээний сүлжээ, графууд дээр хэрэглэгдэх боломжтой юм.

2. GIRVAN NEWMAN АЛГОРИТМЫН ХУРДНЫ САЙЖРУУЛАЛТ

2.1 Girvan Newman алгоритм буюу GN

Энэ алгоритм нь сүлжээ доторх бүх ирмэгийн betweenness эрэмбийг бодож олоод хамгийн өндөр эрэмбэтэй нэг ирмэгийг устаад дахин давтах ба сүлжээнд ямар ч ирмэг үлдээгүй үед зогсоно. Өмнө нь хэлсэнчлэн энэ үйлдэл нь маш өндөр өртөгтэй үйлдэл юм.

Алгоритм 1 GN алгоритм

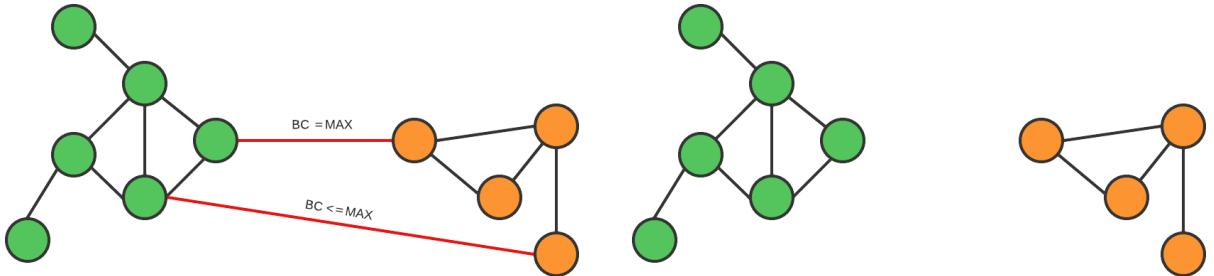
```
1: function girvanNewman( $G$ ) ▷ G–дамжуулж буй граф
2:   while  $edgeNumber$  is not 0 do
3:      $edges \leftarrow calculateBetweenness(G)$  ▷ max betweenness-тэй ирмэгийг устгана
4:      $removeEdge(edges.\text{max})$ 
5:   end while
6: end function
```

Table 2.1 Girvan Newman алгоритмийн ажиллах хугацаа [5].

Өгөгдөл	Төрөл	Оройн тоо	Ирмэгийн тоо	Betweenness бодох хугацаа	GN ажиллах хугацаа
Граф 1	Чиглэлгүй	500	2551	13сек	0.906 сек
Граф 2	Чиглэлгүй	1000	9894	39сек	8.769 сек
Граф 3	Чиглэлгүй	1500	22515	157сек	31.090 сек

Дээрх хүснэгтээс энэ алгоритм нь оройн тоо 1000 ба ирмэгийн тоо 10000 хүрэхэд л нэг удаа betweenness бодох хугацаа болон алгоритмын биелэх хугацаа хэр урт байгааг харж болно.

2.2 Олон ирмэг устгах



(a) GNM ажиллахаас өмнө граф

(b) GNM ажилласны дараа

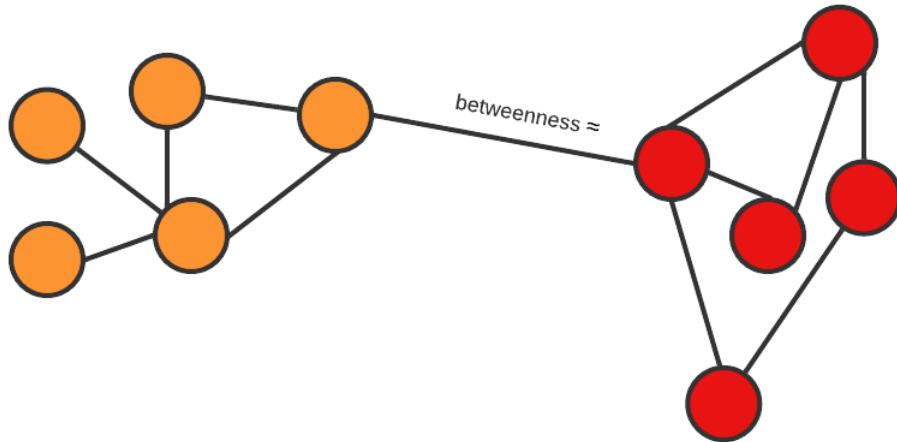
Зураг 2.1 Олон ирмэг устгах / Multiple edge removal

Алгоритм маань хэрхэн өндөр өртөгтэйгээр betweenness эрэмбийг бодоод ердөө ганц ирмэг устган дахин шинээр бодож байгааг өмнөх хэсэгт дурдсан ба хэрэв нэг удаа бодсон эрэмбээ ашиглан тухайн дарааллаар хэд хэдэн ирмэг устгавал ихээхэн цаг хожих боломжтой юм. Учир нь хамгийн өндөр эрэмбэтэй биш ирмэг ч inter-community edge буюу бүлэг хооронд орших ирмэг байх магадлал өндөр байгаа юм. Хэдий бүлэг хоорондын ирмэгээс өөр ирмэг устгасан ч гэсэн зорилтот шинж чанартай ирмэгүүд мөн л устах учир хүлээж авч болох эрсдэл гэж үзэж байгаа юм. Харин нэг удаа бодогдсон эрэмбээр хэдий хэмжээний ирмэг устгах нь бодолцох ёстой чухал зүйл ба хэрэв хэтэрхий олон ирмэг устгавал гүйцэтгэл их хэмжээгээр буурах ба нөгөө талаас хэт бага ирмэг устгавал Girvan Newman алгоритмтай адилхан хугацаанд адилхан үр дүн гаргах юм. [2] Энэхүү санааг доорх алгоритмаар илэрхийлсэн бөгөөд граф доторх ирмэгийн тооны тодорхой хувиар устгаж байгаа болно. Харин хэрэгжүүлэлт хийхдээ нэг удаа устгах ирмэгийн тоог нийт ирмэгийн квадрат язгуураар авсан болно.

Алгоритм 2 GNM (multiple edge removal)

```
1: function girvanNewman( $G$ ) ▷  $G$ -дамжуулж буй граф
2:   while  $edgeNumber$  is not 0 do
3:      $edges \leftarrow calculateBetweenness(G)$ 
4:      $removeEdges(edges, percent)$  ▷ betweenness ирмэгийн дагуу параметрээр
      дамжуулах хувийн тоогоор ирмэгүүдийг устгана
5:   end while
6: end function
```

2.3 Betweenness эрэмбийг ойролцоолох



Зураг 2.2 Ирмэгүүдийн BC эрэмбийг ойролцоолох нь

GN алгоритм нь ирмэгийн BC эрэмбийг бодохдоо боломжит бүх хос оройн богино замыг авч үздэг ба энэхүү замд байгаа ирмэгүүдийн BC эрэмбийг нэмэгдүүлдэг билээ. Тэгвэл энэ процесс нь цаг хугацааны хувьд маш өндөр өртөгтэй ба энэ нь уг алгоритмын гол асуудлуудын нэг юм. Тэгвэл энэхүү BC эрэмбийг яг бодож гаргахгүй ч ойролцоо утгуудыг нь хамаагүй бага хугацаанд олох арга, алгоритмууд сүүлийн жилүүдэд олширч байгаа билээ. Тэдгээрийн нэг төлөөлөгч бол Matteo Riondato тэргүүтэй хүмүүсийн санаачилсан "Fast approximation of

betweenness centrality through sampling” буюу оройн ВС эрэмбийг шилэх байдлаар ойролцоолох арга юм [9]. Энэхүү алгоритмын санаа нь хангалттай тооны (г хэмжигдэхүүн) хос оройг санамсаргүйгээр графаас шилж аван энэхүү хоёр оройн хооронд орших бүх богино замаас мөн нэгийг шилж авах ба энэ замд орших оройн ВС эрэмбийг $1/r$ хэмжээгээр нэмэгдүүлэх явдал юм. Энэхүү г хэмжигдэхүүнүүд нь хэд хэдэн тогтмолууд болох дельта, эпсilon мөн графын диаметрээс хамаарна. Дельта нь алдааны хэмжээг, харин эпсilon нь нэмэгдэхүүн алдааг илэрхийлнэ. Графын диаметрийн хувьд тооцоолол мөн өндөр өртөгтэй ба диаметр ойролцоолох арга алгоритмыг ашиглан утгыг нь гаргаж авч болно. Уг алгоритмд графын диаметрийг ойролцоолоходоо санамсаргүйгээр нэг оройг графаас сонгож авах ба бусад бүх орой руу богино замыг бодож олоод тэдгээр дундаас хамгийн их утгатай хоёр замын уртын нийлбэрээр авч үзэж байгаа юм. Өмнөх хэсэгт дурдсан олон ирмэг устгах санааг ВС ойролцоолох алгоритмтай нийлүүлэн GN алгоритмын хурдаасгах ажлыг хийсэн бөгөөд ингэснээрээ уг алгоритмын хамгийн том сул тал болох гүйцэтгэлийн хугацааг багасгаж байгаа юм.

Алгоритм 3 ВС ойролцоолох алгоритм

Оролт: Граф $G = (V, E)$ $|V| = n$, $\epsilon, \delta \in (0, 1)$

Гаралт: Оройн ВС утгуудыг агуулах жагсаалт

```

1: function bcApprx( $G$ )                                ▷  $G$ -дамжуулж буй график
2:    $b(v) \leftarrow 0$                                      ▷ Бүх ирмэгийн ВС-г 0 болгоно
3:    $VD(G) \leftarrow \text{getVertexDiameter}(G)$            ▷ Диаметр бодно
4:    $r \leftarrow (c/\epsilon^2)(\log_2(VD(G) - 2)) + \ln 1/\delta$  ▷  $r$  хэмжигдэхүүнийг гаргаж авна
5:   for  $i \leftarrow 1$  to  $r$  do
6:      $(u, v) \leftarrow \text{sampleUniformVertexPair}(V)$        ▷ Санамсаргүй хос орой сонгоно
7:      $S_{uv} \leftarrow \text{computeAllShortestPaths}(u, v)$       ▷ Богино замаас санамсаргүй сонгоно
8:      $t \in S_{uv}$                                          ▷  $S_{uv}$  замд орших  $t$  орой эсвэл ирмэг
9:      $b(t) \leftarrow b(t) + 1/r$                            ▷ Орой эсвэл ирмэгийн ВС-Г нэмэгдүүлэх
10:    end for
11: end function

```

3. ХЭРЭГЖҮҮЛЭЛТ, ШИНЖИЛГЭЭ

3.1 Ашигласан өгөгдлүүд

- Karate club: Zachary karate club болох дээд сургуулийн карате клубийн сүлжээ. Zachary гэх хүний карате клубийн хувьд явуулсан судалгаа ба клубийн дотоод үл ойлголцол, маргаанаас болоод Zachary зааварлагч клубийн тал хүнээ авч клубээс гаран шинэ клуб эхлүүлэхдээ клубийн гишүүдийн холбоо зэргийг судалсан ба үр дүнд нь гарсан граф.[10]
- Dolphine dataset: Хэсэг бүлэг делфины харилцаа хамаарлын сүлжээ. [10]
- Facebook food page dataset: Facebook-н итгэмжлэгдсэн хуудаснуудын сүлжээ ба орой нь хуудас, ирмэгүүд нь дундын дагагчдыг илэрхийлнэ.[10]
- Facebook social dataset: Судалгаанд оролцсон бүлэг хүмүүсийн найзуудын жагсаалтаас бүтэх граф [6]
- LastFM dataset: LastFM сошиал сүлжээн дэх ази дахь хэрэглэгчдээс цуглуулсан өгөгдөл бөгөөд орой нь хэрэглэгчид, ирмэг нь тэдгээр хэрэглэгчид дунд орших нийтлэг дагагчдыг илэрхийлнэ.[11]
- Advogato: Advogato сошиал сүлжээн хэрэглэгчдийн хооронд орших холбооны мэдээллийг агуулна.[10]

Сайжруулсан алгоритмыг туршиж үзэхдээ эдгээр өгөгдлүүдийг snap болон network repository сангаас авч туршиж үзсэн ба Karate club, Dolphine dataset өгөгдлүүдийн хувьд ирмэг ба оройн тоо нь 100 дотор багтаж харьцангуй жижиг бүлэг илрүүлэх алгоритмуудын сонгодог жишээ графууд байгаа бол сүүлийн 4 граф нь 600-7000 хүртэлх оройн тоотой, ирмэгийн тоо нь 27,000-88,000 хооронд байгаа дунд хэмжээний граф байна.

Table 3.1 Ашигласан өгөгдлүүдийн мэдээлэл ба snap болон networkrepository-с өгөгдлөө авсан болно.

Өгөгдөл	Төрөл	Оройн тоо	Ирмэгийн тоо	Диаметер
Karate club dataset	Чиглэлгүй	34	78	5
Dolphine dataset	Чиглэлгүй	62	159	8
Facebook food page dataset	Чиглэлгүй	620	27,806	17
Facebook dataset	Чиглэлгүй	4,039	88,234	8
LastFM dataset	Чиглэлгүй	7,624	27,806	15
Advogato dataset	Чиглэлгүй	5,200	47,300	14

3.2 Үр дүн

3.2.1 Жинхэнэ амьдрал дээрх графууд дээр туршисан үр дүн

Дэвшүүлсэн санаагаа дээр дурдсан графууд дээр туршиж үзэн хугацааны биелэлтийг доорх хүснэгтээр илэрхийлсэн байна. GNC алгоритм нь олон ирмэг устгах, BC эрэмбийг ойролцоолох санаануудыг нэгтгэсэн арга ба нэг удаа устгах ирмэгийн хэмжээ нь оройн нийт ирмэгийн тооны квадрат язгуур ба BC эрэмбийг ойролцоолох алгоритмын тогтмолууд болох эпсилон = 0.3 дельта = 0.5 байхаар тус тус авсан болно. GNM нь дан олон ирмэг устгаж байгаа алгоритм ба BC эрэмбийг яг тооцоолж гаргаж байгаа юм. GNM алгоритмын нэг удаа устгах ирмэгийн тоо мөн нийт ирмэгийн тооны квадрат язгуур болно.

Table 3.2 Ажилласан хугацааны үр дүн. GNC нь олон ирмэг устгах мөн ВС ойролцоолох аргыг нийлүүлсэн алгоритм (эпсилон = 0.3, делтар = 0.5 үед). GNM нь олон ирмэг устгах алгоритм. Эхний 3 граф дээр 2 бүлэгт хуваасан бол сүүлийн 3 граф дээр 10 бүлэгт хуваасан

Өгөгдөл	GNC/combined/	GNM/multiple edge/	GN
Karate club dataset	0.00498сек	0.0149сек	0.0368сек
Dolphine dataset	0.03792сек	0.0518сек	0.0738сек
Facebook food page dataset	0.09175сек	14.2979сек	56.6508сек
Facebook dataset	79.3771сек	1356.1541сек	>=18000сек
LastFM dataset	137.0539сек	3618.751сек	>=18000сек
Advogato dataset	41.1248сек	2283.1243сек	>=18000сек

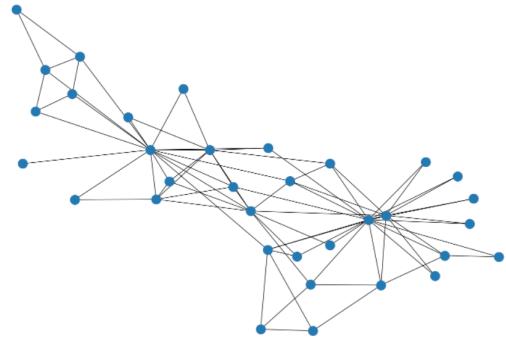
3.2.2 Гүйцэтгэл

GNC алгоритмын гүйцэтгэлийг Girvan Newman алгоритмын буцаах бүлгүүдийн нэгдлийг GNC алгоритмын үр дүнтэй хэдэн хувь таарч байгааг тодорхойлох байдлаар хийсэн болно. Хэдэн хувь таарч байгааг гаргаж авахдаа python хэлний difflib class-н SequenceMatcher аргыг 2 алгоритмын хамгийн өндөр таарч байгаа бүлгүүдийн хувийг авч бүх бүлгүүдийн хувийг дундажлах байдлаар тодорхойлсон.

Table 3.3 Графууд дээр GNC алгоритмын үр дүн GN алгоритмтай хэдэн хувь таарч байгааг харуулж буй гүйцэтгэл. /2 бүлэг/

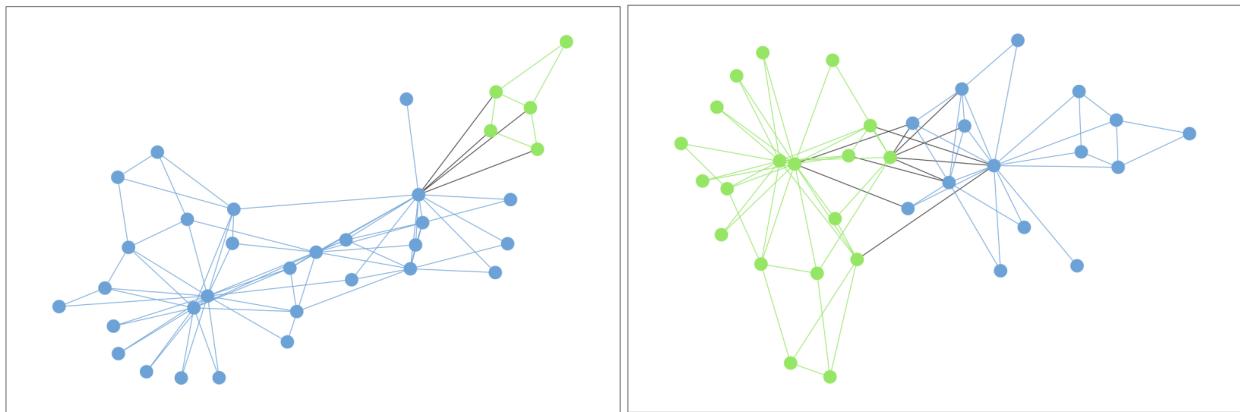
Алгоритм	Karate club	Dolphine dataset	Facebook food page dataset	Facebook dataset
GNC (e=0.1 delta=0.3)	0.95	0.96	0.97	0.73
GNC (e=0.2 delta=0.4)	0.88	0.94	0.91	0.67
GNC (e=0.3 delta=0.5)	0.78	0.90	0.87	0.67

3.2.3 Карате клуб үр дүн



Зураг 3.1 Карате клубын сүлжээ

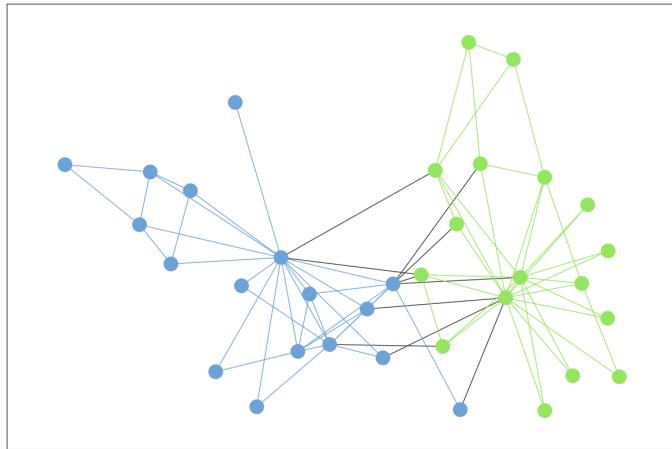
Зураг 3.1 дээр карате клубийн графын ерөнхий дүрслэлийг үзүүлсэн ба Зураг 3.2 дээр энэхүү граф дээр GN алгоритм ба сайжруулалт хийгдсэн GNC алгоритмын үр дүнг харуулсан байна. Энэхүү граф нь хэмжээний хувьд харьцангуй жижигхэн ба сайжруулсан алгоритм маань дутагдалтай ажилласан нь харагдаж байна. Учир гэвэл сайжруулсан алгоритм маань нэг удаа бодогдсон BC эрэмбээр граф дотор орших нийт ирмэгийн тооны квадрат язгуураар ирмэгийг устгаж байгаа болохоор цөөн тооны ирмэг, оройтой график дээр дутагдалтай ажиллаж байгаа юм. Энэхүү графын хувьд нэг удаа устгах ирмэгийн тоо нь 9 юм. Жижиг графуудын хувьд нэг удаа устгаж байгаа ирмэгийн тоог зохих төвшинд багасгавал үр дүн нь сайжрах юм.



(a) GNC алгоритм

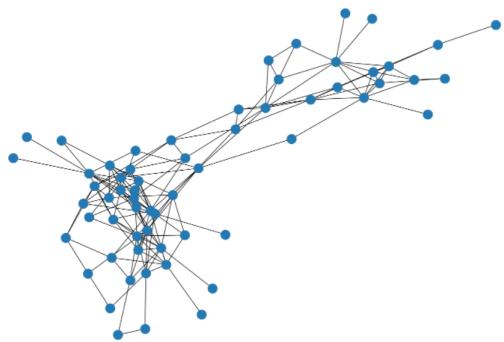
(b) GN алгоритм

Зураг 3.2 Карате клуб үр дүн



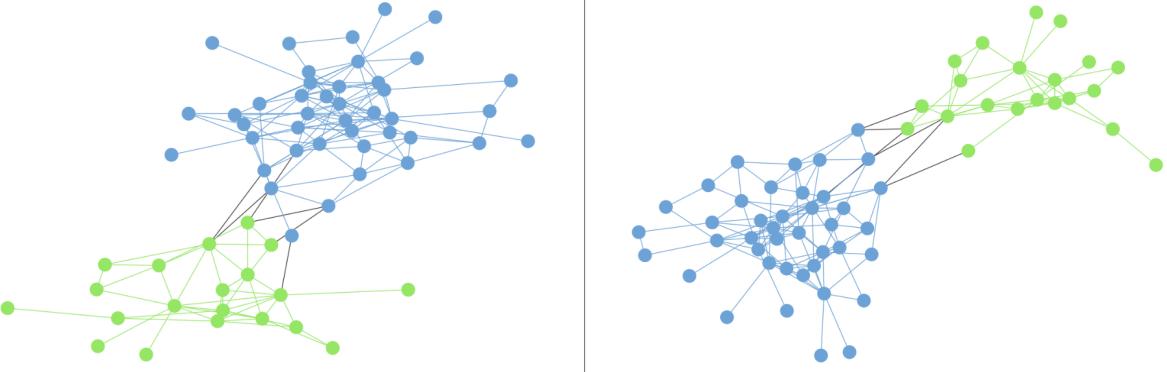
Зураг 3.3 Каратэ клубын үр дүн (нэг удаа устгах ирмэгийн тоог 3 болгож багасгасны дараа.)

3.2.4 *Dolphine* график үр дүн



Зураг 3.4 Хэсэг бүлэг делфины харилцаа хамаарлын сүлжээ

Зураг 3.4 дээр хэсэг бүлэг делфины харилцаа хамаарлын графыг үзүүлсэн ба Зураг 3.5 дээр хурдасгасан алгоритм болох GNC нь GN алгоритмтайгаа дөхсөн үр дүнг энэ удаа үзүүлсэн нь харагдаж байна. Энэхүү графын хувьд өмнөх карате клубтэй харьцуулбал 2 дахин том ба орой болон ирмэгийн тоо хангалттай их үед GNC алгоритмын үр дүн тодорхой хэмжээгээр осөж байна.

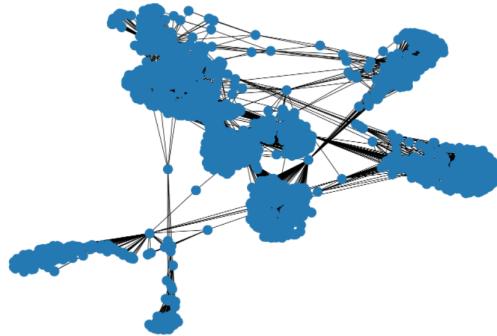


(a) GNC алгоритм

(b) Girvan Newman

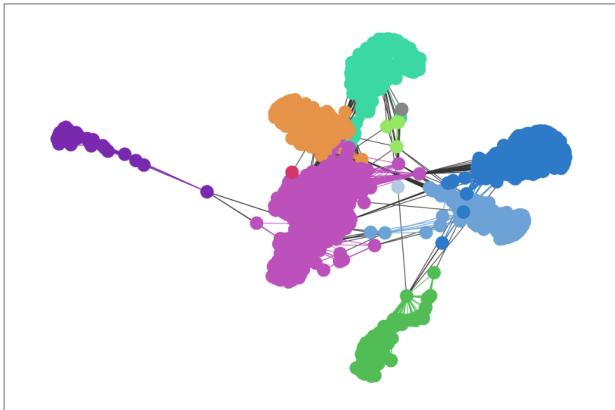
Зураг 3.5 Делфины сүлжээний үр дүн

3.2.5 *Facebook* найзын сүлжээний өгөгдлийн график үр дүн

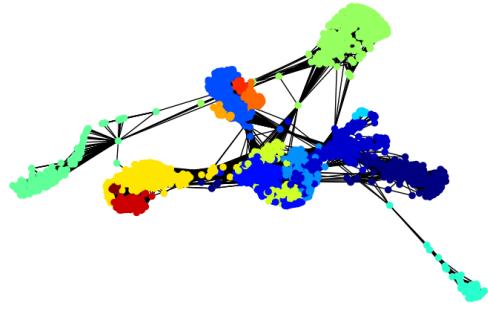


Зураг 3.6 Facebook найзын сүлжээ

Зураг 3.6 дээр facebook-н найзын сүлжээний графыг дүрсэлсэн ба Зураг 3.7 дээр GNC алгоритм ба GN алгоритмуудын үр дүнг дүрсэлсэн байна. Энэ сүлжээ нь өмнөх хэсгүүдэд дүрслэгдсэн сүлжээнүүдээс харьцангуй том ба энэхүү графын оройн тоо 4 мянга орчим, ирмэгийн тоо 80 мянга орчим байгаа билээ. GNC алгоритм нь энэ граф дээр нэг удаа устгах ирмэгийн тоо нь 290 орчим ба GN алгоритмаас бага хугацаанд дөхөх үр дүнг үзүүлж байна.



(a) GNC алгоритм



(b) GN алгоритм

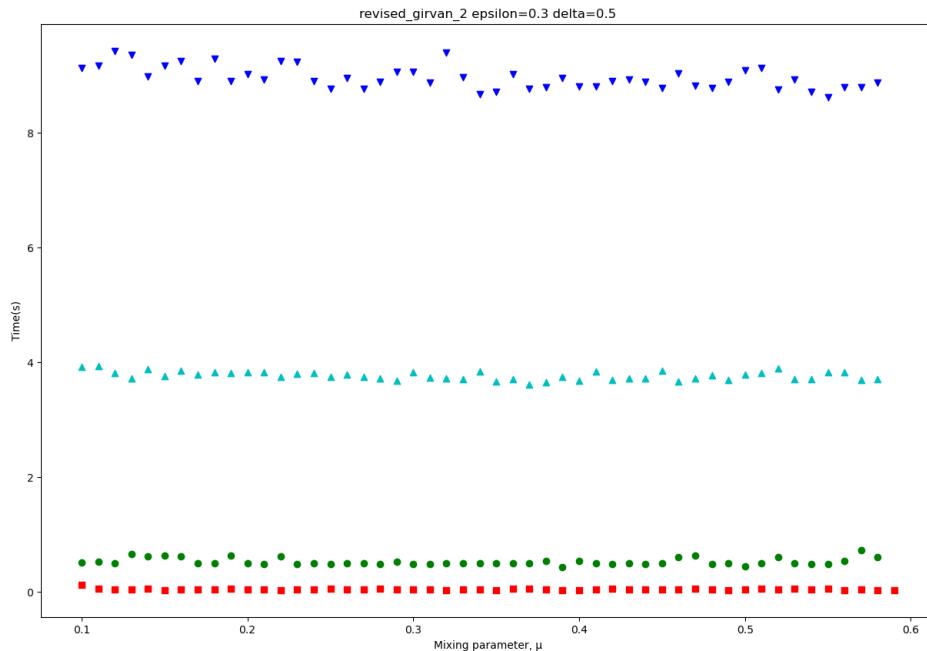
Зураг 3.7 Facebook найзын сүлжээн дээрх алгоритмуудын үр дүн

3.2.6 Хиймэл граф дээр ажиглуулсан үр дүн

GNC алгоритмыг LFR benchmark болох хиймлээр үүсгэсэн сүлжээнүүд дээр туршиж үзсэн бөгөөд mixing parameter болох μ хэмжигдэхүүн, мөн GNC алгоритмуудын тогтмол болох дельта, эпсилон, оройн тоонуудаас хамаарсан хэмжилтүүдийг хийж үзэв. μ хэмжигдэхүүний тодорхойлолт[14]:

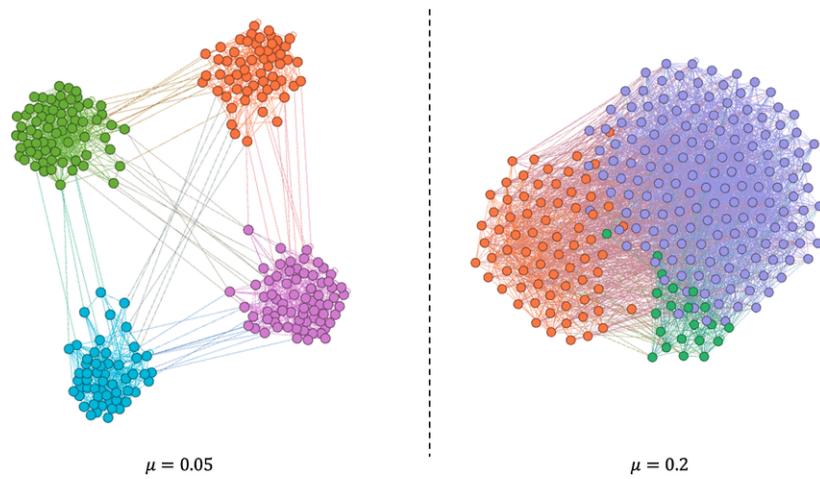
$$\mu = \frac{\sum_i k_i^e}{\sum_i k_i^t}$$

Энд k_i^e нь тухайн оройн бусад бүлэгт харьяалагддаг оройтой холбогдсон ирмэгийн тоо ба k_i^t нь оройн нийт зэрэг(degree) юм. Зэрэг нь тухайн оройд хэдэн ирмэг холбогдсон байгаа тоогоор илэрхийлэгдэнэ.



Зураг 3.8 Хиймлээр үүсгэсэн графын μ тогтмолоос хамаарч өөрчлөгдөх хугацааны биелэлт.
 ▼ - $N = 30000$, ▲ - $N = 15000$, ● - $N = 3000$, ■ - $N = 250$ ба $\epsilon = 0.3$, $\delta = 0.1$ байна.

Хиймлээр үүсгэсэн графын μ хэмжигдэхүүн нь дээр тодорхойлсноор ихсэх тусам граф маань улам шигүү болж бүлэг хоорондын ирмэгүүдийн тоо ихсэж зарим бүлэг илрүүлэх алгоритмууд ажиллах хугацаа нь ихэсдэг ба сайжруулсан алгоритм маань энэ тогтмолын ёгсөлтөөс хамаарч ажиллах хугацаа нь тогтмол байгааг харж болно.



Зураг 3.9 μ тогтмол 0.05 болон 0.2 байхын ялгаа.[12]

ДҮГНЭЛТ

Энэхүү ажилд сүлжээ, бүлэг, ВС эрэмбэ, үүнийг ашиглаж бүлэг илрүүлдэг арга алгоритм, мөн хэрхэн ВС эрэмбийг ойролцоолох тухай судалгаа хийж Girvan Newman алгоритмыг олон ирмэг устгах, ВС эрэмбийг ойролцоолох аргуудаар ажиллах хугацааг нь багасгаж бодит амьдрал дээрх графууд болон хиймэл графууд дээр туршиж, тодорхой үр дүнгүүдийг гаргаж авлаа.

Судалгааны хүрээнд хийгдсэн ажлууд:

1. Сүлжээ, бүлэг илрүүлэх ойлголтуудыг судлав.
2. ВС эрэмбэ, Girvan Newman алгоритмыг судлах хүрээнд тодорхой өгүүллүүдийг уншиж, судалсан.
3. ВС ойролцоолох алгоритмыг судалж хэрэгжүүлэлтийг хийв.
4. Олон ирмэг устгах байдлаар GN алгоритмын хурдыг сайжруулах туршилтыг хийв.
5. ВС ойролцоолох алгоритмаа олон ирмэг устгах санаатай хослуулан хэрэглэж GNC алгоритмын үр дүнг өгөгдлүүд дээр туршиж тодорхой үр дүнгүүдийг гаргаж авсан.
6. GNC буюу хослуулсан алгоритмаа зүгшрүүлэлтийг хийв.

Bibliography

- [1] Linton C. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40(1):35–41, 1977. Publisher: [American Sociological Association, Sage Publications, Inc.].
- [2] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, June 2002.
- [3] Tatsunori B. Hashimoto, Masao Nagasaki, Kaname Kojima, and Satoru Miyano. BFL: a node and edge betweenness based fast layout algorithm for large scale networks. *BMC Bioinformatics*, 10(1):19, January 2009.
- [4] Thamindu Dilshan Jayawickrama. Community Detection Algorithms, February 2021.
- [5] Tripo Matijević, Tijana Vujicic, Jelena Ljucović, Petar Radunović, and Adis Balota. Performance Analysis of Girvan-Newman Algorithm on Different Types of Random Graphs. In *CECIS - 2016*, August 2016.
- [6] Julian McAuley and Jure Leskovec. Learning to discover social circles in ego networks. page 9.
- [7] Mark EJ Newman. Scientific collaboration networks. i. network construction and fundamental results. *Physical review E*, 64(1):016131, 2001.
- [8] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences*, 101(9):2658–2663, March 2004. Publisher: Proceedings of the National Academy of Sciences.
- [9] Matteo Riondato and Evgenios M. Kornaropoulos. Fast approximation of betweenness centrality through sampling. *Data Mining and Knowledge Discovery*, 30(2):438–475, March 2016.
- [10] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.
- [11] Benedek Rozemberczki and Rik Sarkar. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. *arXiv:2005.07959 [cs, stat]*, August 2020. arXiv: 2005.07959.
- [12] Ali Tosyali and Behnam Tavakkol. A node-based index for clustering validation of graph data. *Annals of Operations Research*, 11 2021.

- [13] Jaewon Yang, Julian McAuley, and Jure Leskovec. Community Detection in Networks with Node Attributes. *2013 IEEE 13th International Conference on Data Mining*, pages 1151–1156, December 2013. arXiv: 1401.7267.
- [14] Zhao Yang, René Algesheimer, and Claudio J. Tessone. A Comparative Analysis of Community Detection Algorithms on Artificial Networks. *Scientific Reports*, 6(1):30750, August 2016. Number: 1 Publisher: Nature Publishing Group.

А. КОДЫН ХЭРЭГЖҮҮЛЭЛТ

```
1 #saijruulsan girvan
2 def _without_most_central_edges(G, most_valuable_edge, edge_number):
3     original_num_components = nx.number_connected_components(G)
4     num_new_components = original_num_components
5     limit = round(math.sqrt(G.number_of_edges()))
6     edges = most_valuable_edge(G)
7     while num_new_components <= original_num_components:
8         edges_to_rmv = edges[0:limit]
9         while edges_to_rmv:
10             edge = edges_to_rmv[0]
11             G.remove_edge(*edges_to_rmv[0])
12             if nx.number_connected_components(G) == 2:
13                 new_components = tuple(nx.connected_components(G))
14                 if len(new_components[0])<len(new_components[1]) and
15                     len(new_components[0])<round(math.sqrt(G.
16                         number_of_edges())):
17                     G.add_edge(*edge)
18                 elif len(new_components[1])<=len(new_components[0]) and
19                     len(new_components[1])<round(math.sqrt(G.
20                         number_of_edges())):
21                     G.add_edge(*edge)
22                 else:
23                     break
24             del edges_to_rmv[0]
25             new_components = tuple(nx.connected_components(G))
26             num_new_components = len(new_components)
27             edges = most_valuable_edge(G)
28     return new_components
29 def revised_girvan(G):
30     if G.number_of_edges() == 0:
31         yield tuple(nx.connected_components(G))
32         return
33     # If no function is provided for computing the most valuable edge,
34     # use the edge betweenness centrality.
35     # if most_valuable_edge is None:
36
37     def most_valuable_edge(G):
38         apprx = bcApproxGen(G)
39         return apprx
40
41     # The copy of G here must include the edge weight data.
42     g = G.copy().to_undirected()
43     # Self-loops must be removed because their removal has no effect on
44     # the connected components of the graph.
45     g.remove_edges_from(nx.selfloop_edges(g))
46     edge_number = g.number_of_edges()
47     while g.number_of_edges() > 0:
48         yield _without_most_central_edges(g, most_valuable_edge, g.
```

```

        number_of_edges())

1 def bcApproxGen(G, epsilon=0.1, delta=0.3, universalConstant=0.5):
2     bc = dict()
3     edges = list(G.edges)
4     for edge in edges:
5         bc[edge] = 0
6     vd = (approxVD(G))
7     r = (universalConstant/(epsilon*epsilon)) * (math.log((vd-2),2) +
8         math.log(1/delta))
9     for i in range(round(r+1)):
10        while True:
11            sample_pair = sample(list(G.nodes()), 2)
12            if (nx.has_path(G,sample_pair[0],sample_pair[1])):
13                break
14        shortest_paths = nx.all_shortest_paths(G, source = sample_pair
15            [0], target = sample_pair[1])
16        shortest_paths = list(shortest_paths)
17        sample_path = random.choice(shortest_paths)
18        prev = sample_path[0]
19        for j in range(1,len(sample_path)):
20            try:
21                bc[(sample_path[j],prev)] += 1/r
22            except:
23                bc[(prev,sample_path[j])] += 1/r
24            prev = sample_path[j]
25    bc = dict(sorted(bc.items(), key=lambda item: item[1], reverse=True
26        ))
27    filtered_list = list(bc)
28    return filtered_list

1 #function that approximates the diameter of the graph
2 def approxVD(G):
3     node_number = G.number_of_nodes()
4     random_vertice = random.choice(list(G.nodes()))
5     shortest_paths = nx.single_source_shortest_path(G, random_vertice)
6     ordered_index = sorted(shortest_paths, key = lambda key: len(
7         shortest_paths[key]))
8     last_index = ordered_index[len(ordered_index) - 1]
9     last_index2 = ordered_index[len(ordered_index) - 2]
10    max_shortest1 = shortest_paths.get(last_index)
11    max_shortest2 = shortest_paths.get(last_index2)
12    diameter = len(max_shortest1) + len(max_shortest2)
13    return diameter

1 #Sanguudaa oruulna
2 #saijruulsan girvang fb dataset deer
3 import networkx as nx
4 import numpy as np
5 import glob
6 import os, os.path
7 import math

```

```

8 import collections
9 import re
10 import networkx.algorithms.community as nxcom
11 import time
12 import heapq
13 from matplotlib import pyplot as plt
14 %matplotlib inline
15 plt.rcParams.update(plt.rcParamsDefault)
16 plt.rcParams.update({'figure.figsize': (15, 10)})
17 import random
18 from numpy import random as nprand
19 random.seed(123)
20 nprand.seed(123)

21 #datagaa avah zamaa zaana
22 pathhack = os.path.dirname(os.path.realpath("C:/Users/pc/Desktop/thesis
    /data/dataFb/feature_map.txt"))
23 feat_file_name = "%s/feature_map.txt" % (pathhack,)
24 feature_index = {} #numeric index to name
25 inverted_feature_index = {} #name to numeric index
26 network = nx.Graph()
27 ego_nodes = []

28 #datagiin propertyg unshij avna
29 def load_features():
30     # may need to build the index first
31     if not os.path.exists(feat_file_name):
32         feat_index = {}
33         # build the index from data/*.featnames files
34         featname_files = glob.iglob("%s/data/*.featnames" % (pathhack,))
35         for featname_file_name in featname_files:
36             featname_file = open(featname_file_name, 'r', encoding="utf8")
37             for line in featname_file:
38                 index, name = parse_featname_line(line)
39                 feat_index[index] = name
40             featname_file.close()
41         keys = list(feat_index.keys())
42         keys.sort()
43         out = open(feat_file_name, 'w')
44         for key in keys:
45             out.write("%d %s\n" % (key, feat_index[key]))
46         out.close()

47     global feature_index
48     global inverted_feature_index
49     index_file = open(feat_file_name, 'r')
50     for line in index_file:
51         split = line.strip().split(' ')
52         key = int(split[0])
53         val = split[1]

```

```

57         feature_index[key] = val
58     index_file.close()
59
60     for key in feature_index.keys():
61         val = feature_index[key]
62         inverted_feature_index[val] = key
63     # # for fb
64     ## line bolgonoos featuree yalgaj avna
65     def parse_featname_line(line):
66         line = line[(line.find(' '))+1:] # chop first field
67         split = line.split(';')
68         name = ';' .join(split[:-1]) # feature name
69         index = int(split[-1].split(" ")[-1]) #feature index
70         return index, name
71     #oroiruudaa unshina
72     def load_nodes():
73         assert len(feature_index) > 0, "call load_features() first"
74         global network
75         global ego_nodes
76         ego_nodes = [int(re.search(r'\d+', (x.split("/")[-1].split('.')[0])
77             .group())) for x in glob.glob("%s/data/*.featnames" % (pathhack
78             ,))]
79         node_ids = ego_nodes
80
81         for node_id in node_ids:
82             featname_file = open("%s/data/%d.featnames" % (pathhack, node_id
83                 ), 'r', encoding="utf8")
84             feat_file = open("%s/data/%d.feat" % (pathhack, node_id
85                 ), 'r', encoding="utf8")
86             egofeat_file = open("%s/data/%d.egofeat" % (pathhack, node_id
87                 ), 'r', encoding="utf8")
88             edge_file = open("%s/data/%d.edges" % (pathhack, node_id
89                 ), 'r', encoding="utf8")
90
91             network.add_node(node_id)
92             ego_features = [int(x) for x in egofeat_file.readline().split(
93                 ' ')]
94             i = 0
95             network.nodes[node_id]['features'] = np.zeros(len(feature_index
96                 ))
97             for line in featname_file:
98                 key, val = parse_featname_line(line)
99                 network.nodes[node_id]['features'][key] = ego_features[i] +
100                     1
101                 i += 1
102
103             for line in feat_file:
104                 featname_file.seek(0)
105                 split = [int(x) for x in line.split(' ')]
106                 node_id = split[0]
107                 features = split[1:]
108                 network.add_node(node_id)

```

```

100     network.nodes[node_id]['features'] = np.zeros(len(
101         feature_index))
102     i = 0
103     for line in featname_file:
104         key, val = parse_featname_line(line)
105         network.nodes[node_id]['features'][key] = features[i]
106         i += 1
107
108     featname_file.close()
109     feat_file.close()
110     egofeat_file.close()
111     edge_file.close()
112 #irmeguudee unshina
113 def load_edges():
114     global network
115     assert network.order() > 0, "call load_nodes() first"
116     edge_file = open("%s/facebook_combined.txt" % (pathhack,), "r")
117     for line in edge_file:
118         split = [int(x) for x in line.split(" ")]
119         node_from = split[0]
120         node_to = split[1]
121         network.add_edge(node_from, node_to)
122 #niited ni graphaa unshij avah function
123 def load_network():
124     load_features()
125     load_nodes()
126     load_edges()
127 def feature_matrix():
128     n_nodes = network.number_of_nodes()
129     n_features = len(feature_index)
130
131     X = np.zeros((n_nodes, n_features))
132     for i, node in enumerate(network.nodes()):
133         X[i, :] = network.node[node]['features']
134
135     return X
136 load_network()
137 #zurahad heregleh function
138 def set_node_community(G, communities):
139     '''Add community to node attributes'''
140     for c, v_c in enumerate(communities):
141         for v in v_c:
142             G.nodes[v]['community'] = c + 1
143 def set_edge_community(G):
144     for v, w, in G.edges:
145         if G.nodes[v]['community'] == G.nodes[w]['community']:
146             G.edges[v, w]['community'] = G.nodes[v]['community']
147         else:
148             G.edges[v, w]['community'] = 0
149 def get_color(i, r_off=1, g_off=1, b_off=1):
150     r0, g0, b0 = 0, 0, 0
151     n = 16

```

```

151     low, high = 0.1, 0.9
152     span = high - low
153     r = low + span * (((i + r_off) * 3) % n) / (n - 1)
154     g = low + span * (((i + g_off) * 5) % n) / (n - 1)
155     b = low + span * (((i + b_off) * 7) % n) / (n - 1)
156     return (r, g, b)
157 #zurah function
158 def draw_community(G, communities):
159     plt.rcParams.update(plt.rcParamsDefault)
160     plt.rcParams.update({'figure.figsize': (15, 10)})
161     # Set node and edge communities
162     set_node_community(G, communities)
163     set_edge_community(G)
164     # bulgiin oroin ungiig zaaj ugnu
165     node_color = [get_color(G.nodes[v]['community']) for v in G.
166                   nodes]
167     # bulgiin irmegiiin ungiig zaaj ugnu
168     external = [(v, w) for v, w in G.edges if G.edges[v, w][
169                   'community'] == 0]
170     internal = [(v, w) for v, w in G.edges if G.edges[v, w][
171                   'community'] > 0]
172     internal_color = [get_color(G.edges[e]['community']) for e in
173                       internal]
174     pos = nx.spring_layout(network)
175     nx.draw_networkx(
176         G, pos=pos, node_size=0,
177         edgelist=external, edge_color="#333333", with_labels=False)
178     # zurna
179     nx.draw_networkx(
180         G, pos=pos, node_color=node_color,
181         edgelist=internal, edge_color=internal_color, with_labels=
182             False)
183     plt.show()
184 def drawCommunityInSketches(network, communities):
185     plt.rcParams.update(plt.rcParamsDefault)
186     plt.rcParams.update({'figure.figsize': (15, 10)})
187     plt.style.use('dark_background')
188     set_node_community(network, communities)
189     set_edge_community(network)
190     external = [(v, w) for v, w in network.edges if network.edges[v
191                     , w]['community'] == 0]
192     internal = [(v, w) for v, w in network.edges if network.edges[v
193                     , w]['community'] > 0]
194     internal_color = ["black" for e in internal]
195     node_color = [get_color(network.nodes[v]['community']) for v in
196                   network.nodes]
197     nx.draw_networkx(
198         network,
199         pos=pos,
200         node_size=0,
201         edgelist=external,
202         edge_color="silver",

```

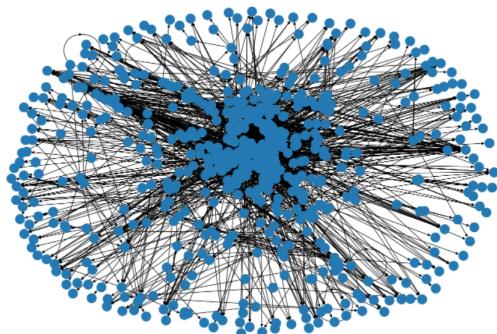
```

195         node_color=node_color,
196         alpha=0.2,
197         with_labels=False)
198     nx.draw_networkx(
199         network, pos=pos,
200         edgelist=internal,
201         edge_color=internal_color,
202         node_color=node_color,
203         alpha=0.05,
204         with_labels=False)
205     plt.show()
206 #girvanaa ajilluulj hugatsaag second aar hevlene
207 nx.draw(network)
208 plt.show()
209
210
211 #1iteration 160 sec orchim
212 start_time1 = time.time()
213 result = revised_girvan(network)
214 for i in range(11):
215     communities = next(result)
216 print((time.time() - start_time1))
217
218
219 pos = nx.spring_layout(network, k=0.1)
220 plt.rcParams.update({'figure.figsize': (15, 10)})
221 nx.draw_networkx(
222     network,
223     pos=pos,
224     node_size=0,
225     edge_color="#444444",
226     alpha=0.05,
227     with_labels=False)
228 plt.show()
229 draw_community(network, communities)
230 drawCommunityInSketches(network, communities)

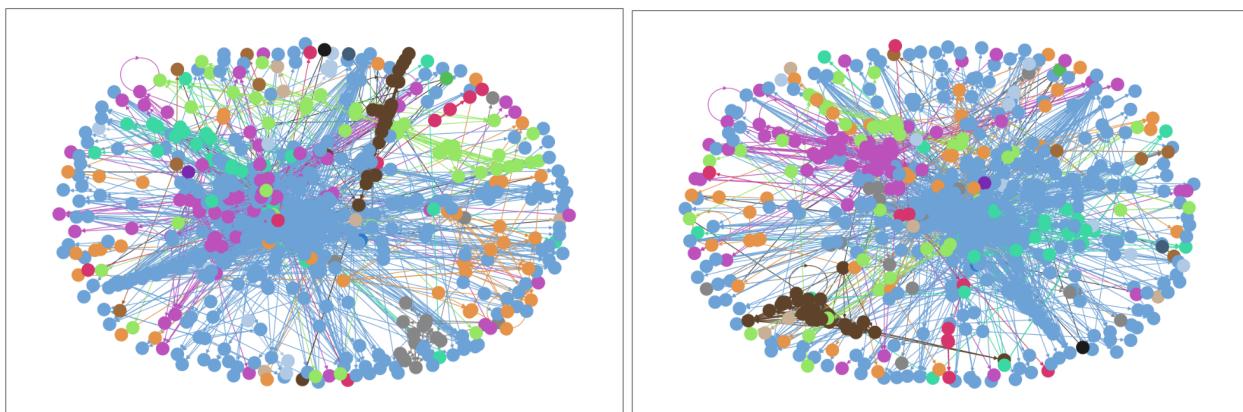
```

В. БУСАД ГРАФИК ҮР ДҮНГҮҮД

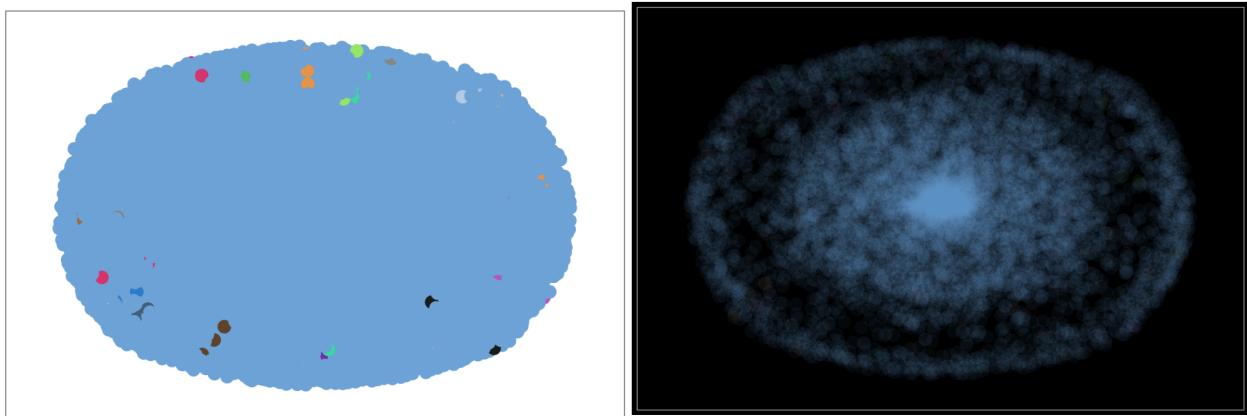
B.0.1 Facebook page график үр дүн



Зураг В.1 Facebook page dataset



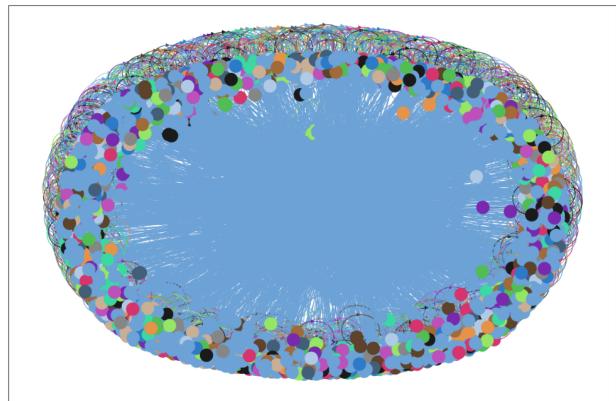
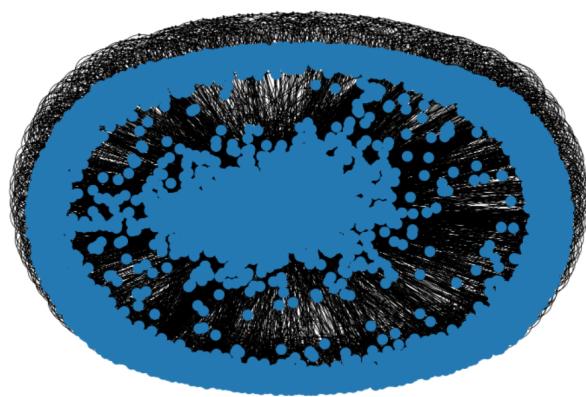
B.0.2 LastFM үграфик үр дүн



(a) Хурдасгасан Girvan Newman

Зураг B.3 LastFm dataset

B.0.3 Advogado үграфик үр дүн



(a) Хурдасгасан Girvan Newman

Зураг B.4 Advogado dataset