Journal of Universal Computer Science, vol. 19, no. 3 (2013), 428-456
ирүүлсэн: 31/3/12, Зөвшөөрөгдсөн: 28/1/13, Хэвлэгдсэн: 1/2/13 @ J.UCS

Граф кластерчлалын нэгэн хувилбарт программчлалын таван хэлийг харьцуулсан нь

Martin Stein
(Karlsruhe Institute of Technology, Germany
mail@martinstein.net)

Andreas Geyer-Schulz
(Karlsruhe Institute of Technology, Germany
andreas.geyer-schulz@kit.edu)

Хураангуй
Орчин үед нийгмийн сүлжээний /social networks/ хэрэгцээ өсөн нэмэгдэж үр ашиг сайтай нийгмийн вэб үйлчилгээнүүдэд зориулсан, гүйцэтгэл нь ихээхэн ашиг авчирдаг хурдтай граф кластерчлалын алгоритмууд /fast graph clustering algorithms/ эрэлт хэрэгцээтэй байна. Программчлалын хэл сонгоход ажиллах үеийн гүйцэтгэл, кодын хэмжээ, хялбархан байх, бусад хэрэгслүүдийг дэмждэг байдал гэх мэт олон үзүүлэлтүүдийг харах хэрэгтэй. Энэхүү өгүүлэлд граф кластерчлалын даалгаварыг хэрэгжүүлэх болон гүйцэтгэх байдлаар нь C++, Java, C#, F#, Python хэлнүүдийг харьцуулсан судалгааны үр дүнг танилцуулна. Бидний хийсэн харьцуулалт нь ерөнхийдөө программ хангамж хөгжүүлэхэд хэрэгтэй чухал ойлголтуудыг өгнө. Өгүүлэлд программчлалын хэлнүүдийг хэдэн онцлог шинжүүдээр нь харьцуулж тоймлон хүргэнэ. Өмнө дурдсан ажиллах үеийн хурд, санах ойн ашиглалт, кодын хэмжээний талаар тайлбарлаад дараа нь эдгээр үзүүлэлтүүдээр хийсэн туршилтын үр дүнг дэлгэрэнгүй тайлбарласан. Туршилтын үр дүнгээс харвал C++ хэл хамгийн хурдтай, санах ойн ашиглалт сайтай байх ба зарим үзүүлэлтээр Java, C#, F# хэлнүүд түүнд ойрхон байдаг байна. Орчин үеийн хэлүүд болох Python, F# хэлнүүдийг ашигласнаар кодын хэмжээг ихээхэн бууруулж болохыг харуулна.

Түлхүүр үгс: Жишиг харьцуулалт, Программчлалын хэлнүүд, хэлний гүйцэтгэл, граф кластерчлал, Модульчлал

1. Удиртгал

Программчлалын хэлнүүдийн тухай хэлэлцүүлэг нь олон хүчтэй үзэл санаанууд дээр тулгуурлан явагддаг хэцүү сэдэв байдаг. Хувийн үзэл бодол, өмнөх туршлагаас бодит харьцуулалтыг гаргаж ирэхэд хэцүү. Хэлнүүдийг харьцуулахдаа ижил төстэй нөхцөл байдалд судлан тогтоох нь хамгийн зөв юм. Амьдрал дээр олон салбарт мэдээллийн технологи нэвтэрч, интернетийн хэрэглээ өсөн нэмэгдэхийн зэрэгцээ өгөгдлийн хэмжээ нэмэгдэж, илүү өргөн цар хүрээг хамарсан боловсруулалт хийгддэг болж байна. Сүүлийн 10 жилд олон тооны нийгмийн сүлээний /Social/ сайтууд гарч ирсэн бөгөөд хэрэглээ нь маш их нэмэгдсэн.
Иймээс граф өгөгдлийг шинжлэх нь чухал бөгөөд энэ тал дээр граф кластер онол чухал үүрэг гүйцэтгэнэ. Шинэ аргуудын хэрэгжилт нь түргэн ажиллагаатай интернет орчинг бий болгоно. Энэ өнцгөөс харвал программчлалын хэл зөв сонгох асуудал чухал юм. Программын ажиллах үеийн гүйцэтгэл, санах ойн ашиглалт нь шууд бус энерги зарцуулалтад нөлөөлдөг. Хичнээн хэмжээний код бичээд ямар үр дүнд хүрэх вэ гэдэг асуултын хариуг олно. Энэ нь программистийн бүтээмж ба хөгжүүлэлтийн зардалд нөлөөлдөг. Программ хангамжийн үйлдвэрлэлд нэг чухал асуудал нь ашиглаж буй программчлалын хэл юм. Тохирсон зөв шийдэлд хүрэхийн тулд: Код нь хөрвүүлэгч программаар шууд ажилладаг байх хэрэгтэй юу? Эсвэл цаг хугацаа шаардсан хөрвүүлэх болон холбох алхам хэрэгтэй юу? гэсэн асуудал гарна. Хэрэгсэл дэмждэг байдал, хөгжүүлэх орчин нь программчлалын хэл бүрийн хувьд ялгаатай байдаг. Тухайлбал IDE / integrated development environments / -үүдийн /Visual Studio, Eclipse гэх мэт/ хэрэглээ нь ч өөр өөр байна.

Энэхүү өгүүллийн гол зорилго нь C++, Java, C#, F#, Python программчлалын хэлнүүдийг нэгэн томоохон тооцооллын хувьд харьцуулан шалгаж, ажиллах үеийн гүйцэтгэл, санах ойн ашиглалт, кодын хэмжээг тодорхойлох юм. Тиймээс бид эхлээд гүйцэтгэх тооцоолол бүхий даалгаварыг тогтоосон. Граф кластерчлалын алгоритмуудын хүрээнд программчлалын олон асуудлыг багтаасан өргөн хүрээтэй харьцуулалт хийнэ. Сонгосон хэлнүүд хэд хэдэн зүйлээр ялгаатай байна: Статик болон динамик төрөлтэй, compile болон interpret хийгддэг, объект хандлагат болон функционал хандлагат гэх мэт.
Энэ өгүүлэлд дээрх хэлнүүдийн абсолют бичлэгийн дүрмийг ойлгуулахыг зориогүй. Гэхдээ энэ асуудал дээр бодитой хэвлэгдсэн өгүүлэл байхгүй байгаа ба бид асуудалд чухал мэдээллээр хангах болно.

2. Судлагдсан байдал

Программчлалын хэлний харьцуулалтын талаар орчин үеийн хэд хэдэн ШУ-ы өгүүлэл хэвлэгдсэн байдаг. 2000 онд Lutz Prechelt 7 хэлийг харьцуулсан судалгаа байдаг. (C, C++, Java, Perl, Python, Rexx, Tcl). Cesarini нар “IMAP client сангуудын 5 хэл дээрх хэрэгжилтийг харьцуулсан нь / Erlang, C#, Java, Python, Ruby /”
Тайлбар: *Энэ хэсэг дутуу орчуулагдсан*

3. Программчлалын хэлнүүдийн тухай

Программчлалын хэлүүдийг бичлэгийн дүрмээс гадна хэд хэдэн шинж чанараар ялгаж болно.

3.1 Ажиллах зарчим /Execution strategy/

Source code нь машин дээр compile хийгдэж эсвэл Interpret хийгдэх гэсэн 2 янзаар ажилладаг. Compiler нь программын кодыг машины кодруу хөрвүүлээд дараа нь процессор дээр ажиллуулдаг. Процедур нь бол эхлээд бичигдсэн кодыг эхнээс нь төгсгөл хүртэл уншиж задлан ялгаад, дүрмийн алдааг хянаж, завсрын дүрслэл гаргаж авна. Дараа нь ассемблер эсвэл машины кодруу хөрвүүлнэ.C++ хэл compiler ашигладаг.
Interpreter нь кодыг алхам алхамаар ажиллуулдаг. Кодын мөр бүр нь задлан ялгагдаж, ажиллах үед машины кодруу хөрвүүлэгдэнэ. Interpreter-ийн ажиллах зарчим нь эхлээд программын кодыг завсрын байт кодруу шилжүүлнэ. Дараа нь программ ажиллах үед тусгай хөрвүүлэгч байт кодыг машины кодруу шилжүүлж өгнө. Python, C#, Java, F# хэлүүд interpreter ашигладаг.

3.2 Төрлийн систем

Өгөгдлийн төрөл заадаг ба төрөл заадаггүй хэлүүд байдаг. Төрөл заадаг хэлүүд статик төрлийг шалгадаг бол төрөл заадаггүй хэлүүд динамик төрлийг шалгадаг. Зарим хэл өгөгдлийн төрөл зарлахыг шаарддаг байхад зарим нь шаарддаггүй. Статик төрөл нь C++, Java, C#, F# хэлүүдэд ашиглагддаг. Python бол өгөгдлийн төрөл зарлахыг шаардахгүй бөгөөд хувьсагчийн төрөл динамикаар шалгагддаг. Cython бол өгөгдлийн төрлийн хувьд эрлийз юм. C шиг төрөл зарлах боломжтой. Динамикаар шалгадаг хэлүүд нь үйлдлүүдийн аюулгүй ажиллагааг программ ажиллах үед шалгадаг учир ажиллах хугацааг нэмэгдүүлдэг. Статик төрөлтэй хэлүүд ажиллах хурд сайтай байдаг. Динамик төрөлтэй хэлүүд илүү уян хатан, сул тал нь алдааг программ ажиллах үед баридаг.

3.3 Программчлалын парадигм /Programming Paradigm/

Орчин үеийн олонх программчлалын хэлүүд объект хандлагат технологитой. Энд өгөгдлийн битүүмжлэл, удамшил, олон хэлбэршил гэсэн ойлголтуудыг онцолно. C++, Java, C# нь объект хандлагат программчлалын хэлүүд мөн.
Функциональ программчлал гэж байх ба түүний эхлэл нь математик юи. Үндсэн үйлдлүүд нь математик функцийн зарчимаар ажиллана. F# хэл нь функционал программчлалын хэл мөн. Python ба Cython нь дээрх ангилалуудад алинд нь ч оруулахад хэцүү.

3.4 Женерик /Generics/

Статик төрөлтэй хэлүүд нь тогтсон олон төрөлтэй байдаг. Энд холбоост жагсаалт, өөрчлөгдөх жагсаалт буюу hash хүснэгт гэх мэт агуулагч классууд ч хамаарна. Бүхэл төрлийн элементтэй, бодит төрлийн элементтэй байхаас үл хамаарч нэг л холбоост жагсаалтыг шууд ашиглах боломжтой байна. Ерөнхий төрөлтэй хэлүүд нь ерөнхий төрлийн өгөгдүүдтэй болон параметрүүдтай функц болон классуудыг бичих боломж олгодог. Хатуу төрлүүдийг нэгээс нөгөөд хөрвүүлэхэд тухайлбал тэмдэгт мөрийг бүхэл тоо болгож хөрвүүлэхэд hash table ашиглана. C++ хэлний хувьд үүнийг template буюу загвар функцээр

илэрхийлнэ. Төрөл хөрвүүлэлт нь compile time үед хийгддэг. C#, F# хэлнүүдийн хувьд төрөл хөрвүүлэлт нь CLR орчинд программ ажиллах үед /runtime/ хийгддэг.

	Execution strategy (3.1)	Type system (3.2)	Main paradigm (3.3)	Generics (3.4)
C++	compiled	static	OOP	compile-time
Java	JIT-compiled	static	OOP	type-erasure
C#	JIT-compiled	static	OOP	runtime
F#	JIT-compiled	static	functional	runtime
Python	interpreted	dynamic	both	n/a
Cython	hybrid	hybrid	both	compile-time

Table 1: Overview of programming language characteristics

4. Харьцуулалт хийх тохиргоо / The Setup of the Comparison /

Энэ өгүүллийн гол зорилго нь нийгмийн сүлжээний үйлчилгээнд хамааралтай граф кластерчлалын даалгаварын хувьд программчлалын хэлнүүдийг харьцуулан хүргэх юм.

4.1 Хамрах хүрээ/Scope and Complexity/

Фибаночийн тоон дарааллыг тооцоолох бяцхан даалгавараар хэлүүдийг түргэн харьцуулж болох ч тэр нь бодит амьдрал дахь хэрэглээг бүрэн илэрхийлж чадахгүй. Иймээс томоохон программын хувьд асуудлыг судлана.

4.2 Программчлалын хүрээ /The Programming Domain/

Харьцуулах даалгавар маань граф кластерчлалын талбар дээр тодорхойлогдоно. Графын нэг сонгодог жишээ нь нийгмийн сүлжээ юм. Хүн бүр нэг орой буюу зангилаа болно. Хүмүүсийн хоорондох найзын харгалзаа нь ирмэг болно.

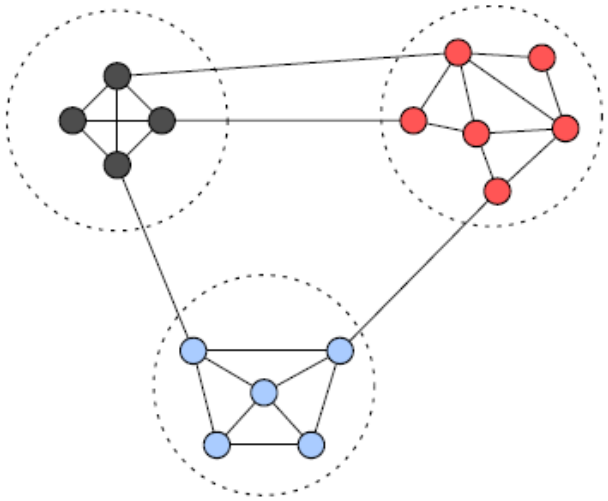

Figure 1: A graph containing three clusters

4.3 Модульчлал /Modularity/

Хар ухаанаар бодоход сайн кластерчлал нь кластеруудын хоорондох холбоосуудын жинг хадгалж байхын зэрэгцээ кластер хоорондын ирмэгийн тоог ихээр агуулж байдаг. Гэвч зүгээр л бүх оройнуудыг ижил хэсгүүдэд хуваасан байх шаардлага байна. Newman нарын дэвшүүлсэн модульчлалын хэмжүүр Q –г хэмжих санааг цааш нь баяжуулна.

$\zeta = \{C_1, C_2, \ldots, C_k\}$ нь хоосон биш элементтэй иргэдийн k ширхэг олонлог байг. $e_{i,j}$ нь C_i ба C_j хэсэг дэх оройнуудын хооронд холбосон ирмэгүүдийн тоо, a_i нь C_i хэсэг

дэх оройнуудруу холбогдсон ирмэгүүдийн тоо байхад Q нь дараах байдлаар тодорхойлогдоно.

$$Q = \sum_i (e_{ii} - a_i^2) \qquad (1)$$

Хэмжүүр нь кластерууд доторх ирмэгүүдийн тоог, тухайн кластерт гаднаас ирж буй ирмэгүүдийн тоотой харьцуулдаг. Онолын хувьд Q-ийн хамгийн их утга нь Q=1.0 байдаг. Практик амьдралд Q>0.3 байдаг.

4.4 **Ажлын тодорхойлолт/Task Description/**

Algorithm 1: High-level specification of the clustering task

Input: Undirected graph in Pajek format
Output: List of clusters

```
graph ← new Graph                                  /* adjacency list */
for line in section *edges do
    graph.add_edges_from(line)

algorithm ← select_algo(command-line)              /* strategy pattern */
joinpath ← algorithm(graph)                        /* runtime measured */

max_Q, index ← get_max_join(joinpath)
clusters ← join_up_to(joinpath, max_Q, index)
for cluster in clusters do
    print(cluster)
```

Энэ даалгавар нь Pajek формат бүхий текст файлд хадгалагдсан нэг графыг уншаад түүнийг санах ойд дүрсэлнэ. Зэрэгцээ матриц нь n зангилааны хувьд $O(n^2)$ зай шаардлагатай ба том графын хувьд тохиромжгүй бөгөөд сийрэг графын хувьд тохирно. Энд сонгосон жинлэгдсэн зэрэгцээ жагсаалт нь санах ойд дүрслэх хамгийн сайн арга мөн. Дараа нь 2 кластерчлалын алгоритмын /greedy and randomized greedy/ нэгийг сонгож байна. Эцэст нь үр дүн болох кластерууд файлд хадгалагдана.
Харьцуулалтын гол цөм болох 2 кластерийн алгоритм нь Newman –ий *greedy* процедур [Newman 2004] ба *randomized greedy* арга юм. [Ovelgönne нар. 2010]. Аль ч аргын хувьд эхлээд зангилаанууд нь кластеруудад хуваагдана. Кластерууд нь хоорондоо холбогдох ба үр дүн нь холбох зам байна. Алхам бүрт Q-ийн утга тооцоологдоно. Энэ замын дагуу хамгийн өндөр модультай үр дүнг сонгоно. Greedy алгоритмд боломжит бүх холболтуудын хувьд ΔQ модулийн өөрчлөлтийг тодорхойлох ба хамгийн их өөрчлөлттэй үр дүнг сонгоно. Сийрэг граф дээр алгоритмын ажиллах хугацаа нь n зангилааны хувьд $O(n^2)$ байна. Randomized greedy аргын хувьд боломжит бүх холболтуудыг шинжлэхийн оронд алхам бүрт цөөн к ширхэг кластерийг санамсаргүйгээр сонгоно. Сонгогдсон кластерууд болон тэдгээрийн хөршүүдийн хоорондох холбоосуудыг тооцоолох ба тэдгээрээс хамгийн сайныг нь сонгоно. к тогтмол утгууд нь ($k \leq 2$) ажиллах хугацааг сийрэг сүлжээн дэх дундажаас $O(n \log n)$ хуулиар бууруулна.
Кластеруудыг дүрслэх өгөгдлийн бүтэц чухал юм. Кластерууд болон хуваагдсан хэсгүүдийн хоорондох холбоосийн ирмэг, жингүүдийг хадгалахад хэрэгтэй. n зангилааны хувьд $O(n^2)$ зай хэрэгтэй учир ердийн матриц ашиглах боломжгүй юм. Кластерууд нь бүхэл утгуудыг бодит утгуудруу буулгах hashtable-уудын массив байдлаар дүрслэгдэнэ. Массив дахь нэг hashtable нь нэг кластерийг илэрхийлнэ. Бүхэл тоон утгууд нь кластеруудын индексийг заах бол бодит утгууд нь оройнуудын хоорондох бүх ирмэг жингүүдийн нийлбэрийг заана.
Кластерчлал нь 300 мянгаас олон оройтой, 1 саяаас олон ирмэгтэй олон томоохон графууд дээр хийгддэг.

Algorithm 2: Randomized greedy modularity clustering

Input: *graph* - in adjacency list representation
Output: *joinpath*

```
e ← new Matrix(graph)                          /* array of hash tables */
a ← new Array(graph.size)
for row i in e do
    a_i ← row_sum(e_i)
for i in 1 to size(e) do
    max_ΔQ ← −∞
    selected ← k_random_clusters(e)
    for cluster c in selected do
        for neighbors n of c do
            ΔQ ← 2(e_cn − a_c a_n)
            if ΔQ > max_ΔQ then
                max_ΔQ ← ΔQ
                best join ← (c,n)
    perform_join(best join)
    joinpath ← joinpath + best join
return joinpath
```

Иймээс тухайн асуудал хэд хэдэн ойлголтуудыг хамрана.

1. Оролт гаралтын файлтай ажиллаж задлан ялгал хийх.
2. Боломжит томоохон мэдээллийн сангууд үүсгэх
3. Стратеги загвар хэрэгжүүлэх /Кластерчлалын арга нь программ ажиллах үед сонгогдоно/
4. Цуглуулга /collection/ ашиглах

5. Үнэлгээ /Evaluation/

Энэ хэсэгт туршилтаас цуглуулсан үр дүнгүүдээс үнэлгээ хийнэ.

5.1 Харьцуулалт /Comparability/

Код нь нэг аргаар бичигдсэн байх хэрэгтэй. Кодын хэмжээ, ажиллах үеийн гүйцэтгэл гэх мэт программын хэмжигдэхүүнүүд нь туршлага, хэрэлсэл дэмждэг байдал, цаг зарцуулалт гэх мэт хэд хэдэн хүчин зүйлээс хамаардаг. Программистуудын ур чадвар өөр өөр байдаг тул үр дүнд нөлөөлнө. Тэгэхээр даалгаварыг өөр өөр хэл дээр өөр өөр хүмүүсээр хийлгэх нь асуудалтай юм. Иймээс зохиогч нар randomized greedy алгоритмыг бүх хэл дээр бичсэн. Тэр программчлалын бүх хэлнүүдийн мэдлэгтэй, java ба python хэлүүд дээр 2 жилийн туршлагатай. Бүх кодууд ижил туршлагаар бичигдсэн тул харьцуулах боломжтой.

Бас нэг асуудал нь оролт гаралтын өгөгдлийн формат тодорхой байх хэрэгтэй. Эцэст нь хэлэхэд хэмжээ нь өөрчлөгддөг массивууд, hashtable гэх мэт өгөгдлийн бүтцүүд нь харьцуулалтын гол цөм нь болно. Хэл бүр тийм өгөгдлийн бүтцүүдийг агуулсан сангуудыг тодорхойлсон байдаг. Тухайлбал C++ хэлэнд STL/Standard template library/ сан байдаг бол C# хэлэнд System.collections namespace байдаг.

Бид программчлалын хэлүүд дэх алгоритмын ажиллагааг харьцуулах, өндөр чанартай код гаргаж авах практик туршлага болон нийтлэг санаануудыг судлахад ихээхэн цаг зарцуулсан. Код бичих цагийг үнэлээгүй бөгөөд алгоритмуудыг программчлалын хэлүүдэд буулгаж бичихэд 4 сар орчим хугацаа зарцуулсан.

5.2 Хэмжээс /Metrics/

Программчлалын хэлүүдийн илэрхийлэх чадвар нь ихээхэн ялгаатай байна. Нэг хэл асуудлыг кодын 1 мөрөөр товч тодорхой илэрхийлж болж байхад нөгөө нь олон мөр кодоор тухайн асуудлыг шийддэг байж болно. Кодын хэмжээ бол программ хөгжүүлэлтэд чухал

хэмжээс болно. Программын хэмжээг кодын мөрийн тоогоор /Source line of code SLOC/ тодорхойлно. Кодын физик мөрийн тоо гэдэг нь программ дахь бүх мөрийн тоог хэлнэ. Энд тайлбар болгосон мөрүүд ч орно. Логик мөрийн тоо гэж тайлбар болгосон мөрүүдээс бусад бүх мөрийн тоог хэлнэ. Бид хүчинтэй кодын мөрүүд /effective lines of code eLOC/ гэж нэрлэх хэмжүүрийг ашиглав.Энэ нь хоосон мөрүүд, тайлбар мөрүүдийг оруулалгүй тооцсон мөрүүдийн тоог хэлнэ. Яг ижил үйлдлийг java болон python хэл дээр бичвэл:

Java:

```
Map<String, Integer> wordCount = new HashMap<String, Integer>();
```

Python:

```
word_count = {}
```

байна. Энэ мөрүүд аль нь ч 1 eLOC болно. Бичлэгийн урт эрс ялгаатай байна. Тиймээс бид тайлбар мөрүүдийг хасаж тооцсон, дараалсан хоосон зайнуудыг нэг зайгаар тооцсон кодын хэмжээг байтаар илэрхийлсэн eByte /effective Byte / гэж нэрлэх хэмжүүрийг ашигласан. Хувьсагч болон классын нэрийг хэл бүрт адилхан тодорхойлж ашигласан.

Кодын хэмжээнээс гадна шийдлүүдийн гүйцэтгэл их сонирхол татна.Граф өгөгдлүүд нь өгөгдлийн санд эсвэл текст файлд хадгалагдсан байх ба графыг уншиж хугацааг хэмжих нь чухал. Иймээс бид ажиллах хугацааг бодит цагаар болон CPU цагаар тодорхойлно.

Мөн хэлнүүдийн санах ойн зарцуулалтыг хэмжих нь чухал юм. Бид бүх даалгаварын хувьд хамгийн оргил санах ойн ачааллыг хэмжинэ.

5.3 **Өгөгдлийн олонлогууд /Datasets/**

Бид бодит амьдрал дээрх 6 өгөгдлийн олонлогийг хэмжээний хувьд өсөх эрэмбээр авч үзсэн. Pajek файлын формат нь чиглэлтэй болон чиглэлгүй граф сүлжээнд ашиглагддаг бол кластерчлалын алгоритмууд нь чиглэлгүй графд чиглэсэн байдаг. Тиймээс чиглэлтэй графд шаардлагатай үед тэгш хэмтэй олон ирмэг, битүү гогцоонуудыг устгадаг.

Эхний өгөгдлийн олонлог нь Karate Граф юм. Krate клубын гишүүн болох 34 зангилаанаас бүрдэх бөгөөд тэдний найзын холбоо нь 78 ирмэг дүрсэлдэг.Энэ сүлжээг граф кластерийн алгоритмуудын чанарыг шалгах үндсэн үзүүлэлт болгон ашигладаг.

Хоёр дахь өгөгдлийн олонлог нь Америкийн 115 их дээд сургуулийн хөл бөмбөгийн багууд графын орой болдог, тэдгээрийн хоорондох тоглолтууд нь ирмэг болдог граф юм. Эдгээр багуудын тоглолт нь нийтдээ 613 ирмэг үүсгэсэн байна.

Дараагийн өгөгдлийн олонлог нь Rovira i Virgili их сургуулийн 1133 хүний хооронд и-мэйл солилцох графыг авч үзэх ба энэ нь 5451 чиглэлгүй ирмэгтэй.

Сүүлийн өгөгдлийн олонглог нь nd.edu домэйн хаягтай вэб хуудсуудын олонлогоос бүрдэх ба тэдгээр нь хоорондоо чиглэлтэй сүлжээ үүсгэн холбогдсон байна.

	Vertices	Edges	Density	Reference
karate	34	78	0.13904	[Zachary 1977]
football	115	613	0.09352	[Girvan and Newman 2002]
email	1133	5451	0.00850	[Guimerà et al. 2003]
pgp	10680	24316	0.00043	[Boguñá et al. 2004]
condmat	27519	116181	0.00031	[Newman 2001]
www	325729	1090108	0.00002	[Albert et al. 1999]

Table 2: Dataset overview

5.4 **Туршилтын орчин / Testing Environment**/

Туршилтыг 2.40 GHz хурдтай Intel CoreTM2 Duo P8600 CPU-тэй, 4GB RAM-тай, Windows 7 service pack 1 үйлдлийн систем бүхий компьютерт хийсэн. Процессорын Power settings нь highest performance горимд тохируулагдсан. Нэн чухал биш даалгаваруудыг цуцалж, машиныг сүлжээнээс салгасан. Бүх хэлүүд 32 бит горимд ажиллахаар тохируулсан.

C++ кодыг хөрвүүлэхдээ GCC 4.5.2 хөрвүүлэгчийг, Java кодыг Oracle-ийн HotSpot JVM 1.6.0_25 хувилбараар, C# кодыг хэлний 4,0 хувилбарт үндэслэн .NET framework 4.0.3

хувилбараар, F# кодыг F# 2.0 compiler-аар, Python кодыг стандарт CPython release 2.7.1 хувилбараар, Cython кодыг C++ хөрвүүлэгч ашиглан хөрвүүлсэн. Бүх хэлнүүдийн ажиллагааг нэг thread бүхий горимд туршсан.

5.5 **Ажиллах үеийн үр дүнгүүд /Runtime results/**

Randomized greedy алгоритмыг хэл бүрийн хувьд, dataset бүрийн хувьд тус бүр 100 удаа ажиллуулж туршсан. Greedy алгоритмыг эхний 5 dataset-ийн хувьд 50 удаа, сүүлийн 1 том dataset-ийн хувьд 3 удаа ажиллуулж туршсан.

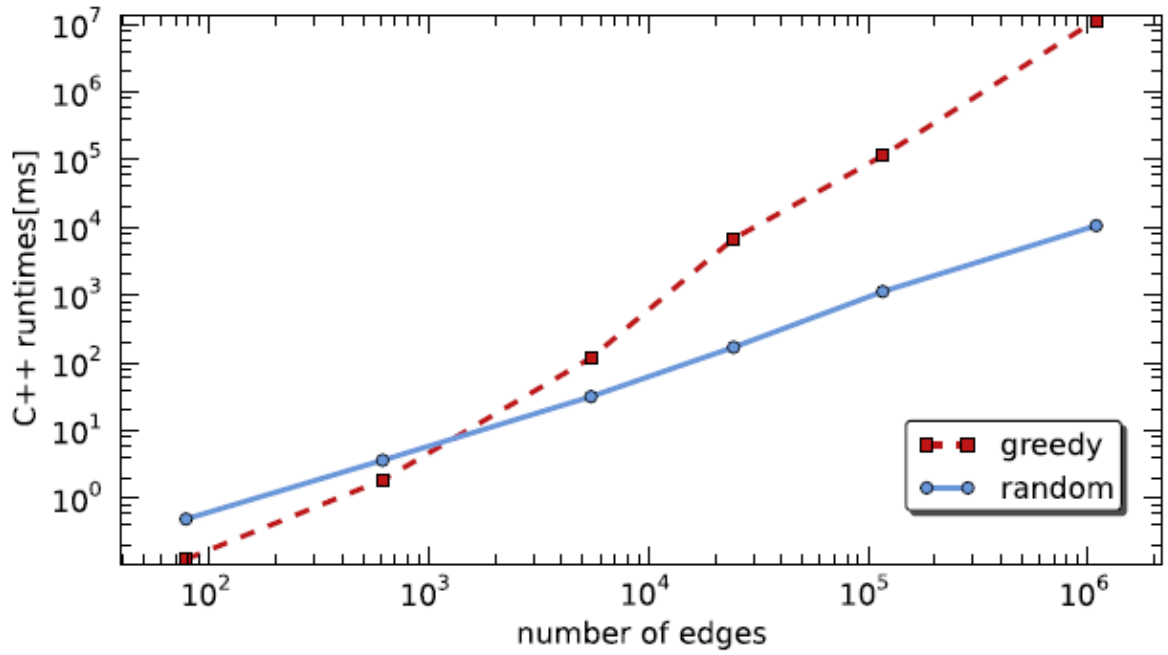

Figure 2: Runtimes[ms] for C++, arithmetic means

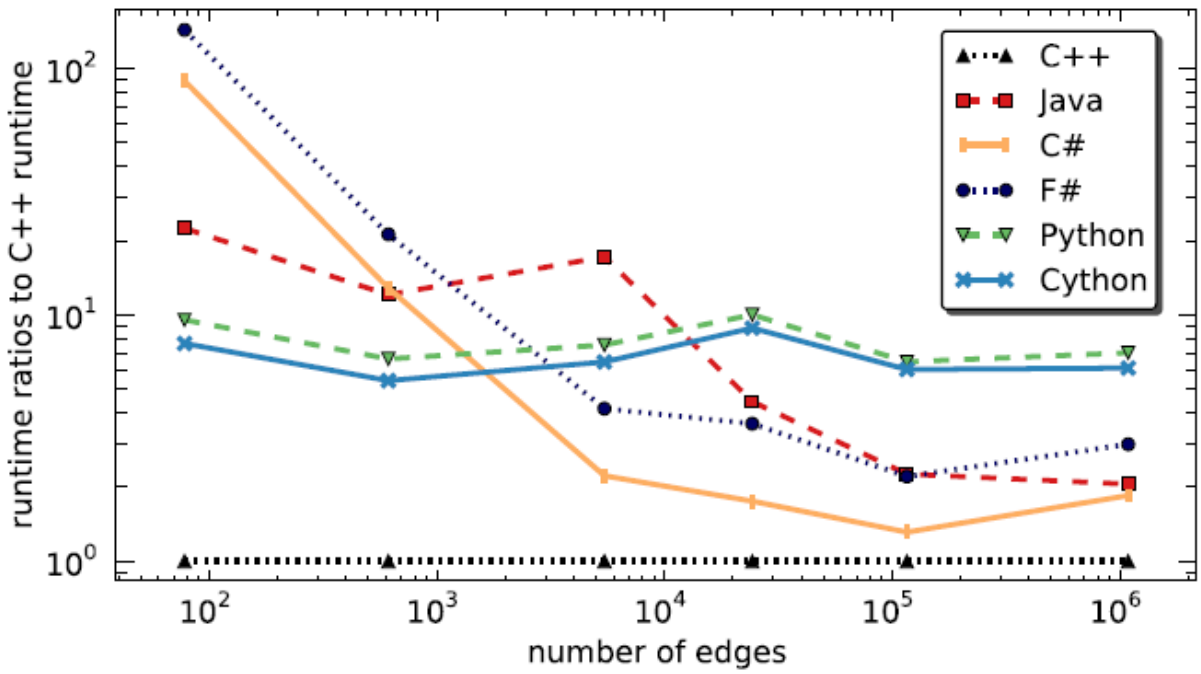

Figure 3: Runtime ratios to C++ times for Randomized Greedy

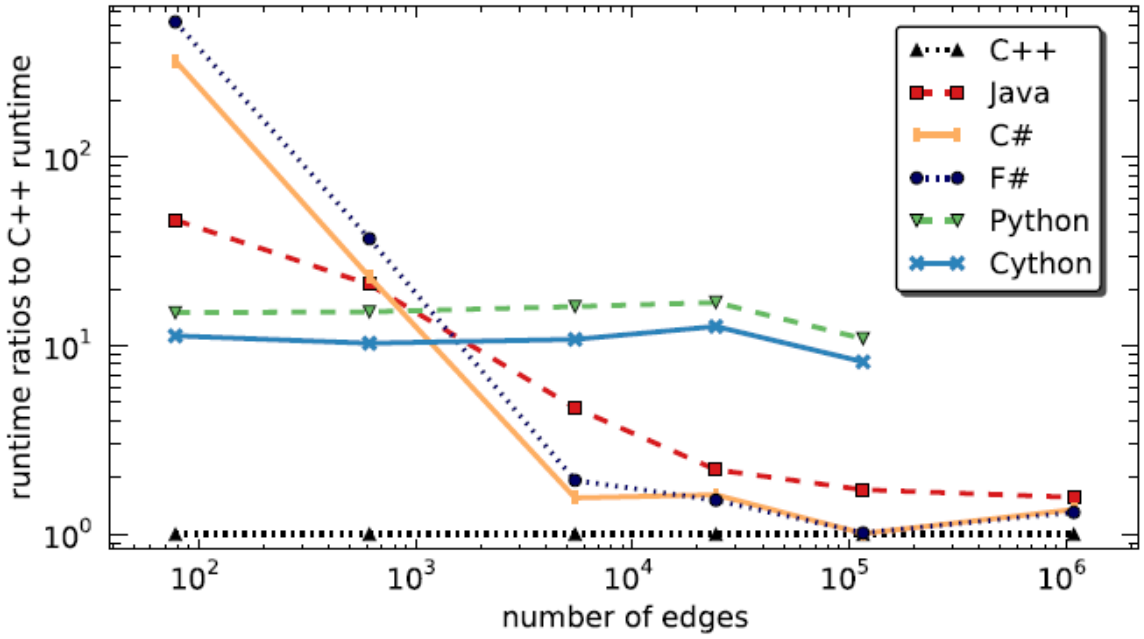

Figure 4: Runtime ratios to C++ times for Newman's Greedy

5.6 **Санах ойн ашиглалт /Memory result/**

Хамгийн том Dataset-ийн хувьд C++ хэл 203.9 MB санах ашиглаж байсан.

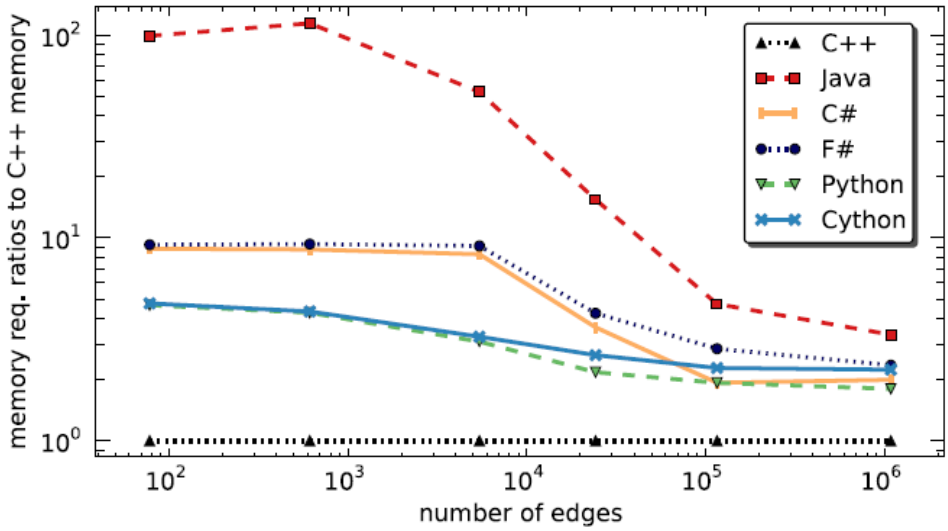

Figure 5: Memory requirement ratios to C++ memory for Randomized Greedy

5.7 **Кодын хэмжээ /Code size result/**

	C++	Java	C#	F#	Python	Cython
eLOC	587	453	376	275	237	243
eBytes	21964	15732	14230	10655	8426	8655
$\frac{eBytes}{eLOC}$	37.4	34.7	37.8	38.7	35.6	35.6

Table 3: Code sizes for each implementation

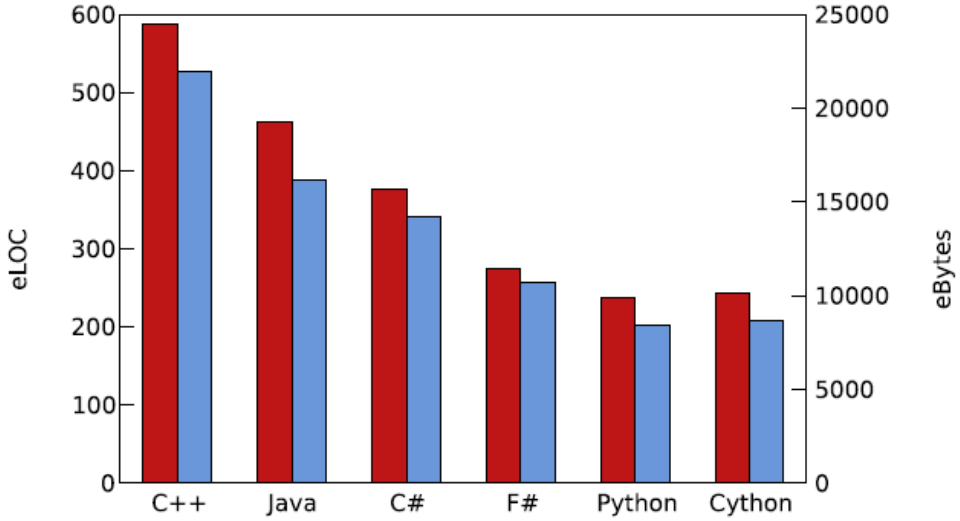

Figure 6: Code sizes in eLOC(left) and eBytes(right)

6. Дүгнэлт

Энэхүү өгүүлэлд граф кластерчлалын нэгэн хувилбарт программчлалын 5 хэл дээр харьцуулсаныг танилцууллаа. Программчлах ур чадварын ялгааг арилгах үүднээс зохиогч өөрөө бүх кодыг бичсэн.

Дараа нь хэмжээгээр өөр өөр 6 Dataset дээр ажиллуулж туршсан. C++ хэл бол хамгийн хурдтай бөгөөд санах ойн ашиглалтаар хамгийн сайн хэл болох нь харагдсан. Гэвч кодын хэмжээгээр хамгийн их байдаг байна. Python хамгийн богино кодтой боловч C++ хэлнээс удаан ажиллагаатай. C#, Java, F# хэлнүүд нь хурд болон кодын хэмжээгээр C++ хэлнээс бага байна. Та бүхэн ирээдүйд программчлалын хэлний харьцуулалтыг өөр олон чиглэлээр өргөтгөх хэрэгтэй.

1. Программчлалын бүтээмж нь ПХ-н инженерчлэлд чухал байр эзэлнэ. Энэ үзүүлэлтээр харьцуулах нь чухал.
2. Олон цөмтэй CPU-ний боломжийг ашиглаж зэрэгцээ боловсруулалтын үед харьцуулалт хийх хэрэгтэй.
3. Дэлхийн хэмжээний том нийгмийн сүлжээний хувьд томоохон граф дээр харьцуулах.

Ашигласан эх сурвалж

[tim 2010] “Timers, Timer Resolution, and Development of Efficient Code”, Technical Report, Microsoft Corporation (2010), http://www.microsoft.com/whdc/system/pnppwr/powermgmt/Timer-Resolution.mspx [accessed 10 June 2011].

[ben 2011] “Computer Language Benchmarks Game”, http://shootout.alioth.debian.org/ (2011), [accessed 31 May 2011].

[sci 2012] “SciViews Benchmark”, http://www.sciviews.org/benchmark/ (2012), [accessed 30 Dec 2012].

[con 2013] “How to: Disable Concurrent Garbage Collection”, http://msdn.microsoft.com/enus/library/at1stbec[accessed 19 Feb 2013].

[hot 2013] “Java SE 6 HotSpot[tm] Virtual Machine Garbage Collection Tuning”, http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html (2013), [accessed 19 Feb 2013].

[Albert et al. 1999] Albert, R., Jeong, H. and Barabási, A.-L., “Internet: Diameter of the World-Wide Web”, Nature, 401, 6749, (1999), 130–131.

[Albrecht and Gaffney 1983] Albrecht, A. and Gaffney, J., J.E., “Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation”, IEEE

Transactions on Software Engineering, SE-9, 6, (1983), 639 – 648, ISSN 0098-5589, doi: 10.1109/TSE.1983.235271.

[Anda et al. 2009] Anda, B., Sjoberg, D. and Mockus, A., “Variability and Reproducibility in Software Engineering: A Study of Four Companies that Developed the Same System”, Software Engineering, IEEE Transactions on, 35, 3, (2009), 407 –429, ISSN 0098-5589, doi: 10.1109/TSE.2008.89.
[Boehm 1981] Boehm, B. W., Software Engineering Economics, Prentice Hall, Englewood Cliffs, NJ (1981).
[Boehm et al. 2000] Boehm, B. W., Clark, Horowitz, Brown, Reifer, Chulani, Madachy, R. and Steece, B., Software Cost Estimation with Cocomo II, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition (2000), ISBN 0130266922.
[Boguñá et al. 2004] Boguñá, M., Pastor-Satorras, R., Díaz-Guilera, A. and Arenas, A., “Models of social networks based on social distance attachment”, Phys. Rev. E, 70, 5, (2004), 056122, doi:10.1103/PhysRevE.70.056122.
[Cardelli 2004] Cardelli, L., “Type Systems”, in A. B. Tucker (editor), The Computer Science and Engineering Handbook, chapter 97, CRC Press (2004).
[Cesarini et al. 2008] Cesarini, F., Pappalardo, V. and Santoro, C., “A comparative evaluation of imperative and functional implementations of the imap protocol”, in Proceedings of the 7th ACMSIGPLAN workshop on ERLANG, ERLANG ’08, ACM, New York, NY, USA (2008), ISBN 978-1-60558-065-4, 29–40, doi:10.1145/1411273.1411279.
[Cormen et al. 2009] Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C., Introduction to Algorithms, The MIT Press, Cambridge, Massachusetts, 3 edition (2009), 589–593.
[Ecma International 2001] Ecma International, ECMA-334: C# Language Specification, Ecma (European Association for Standardizing Information and Communication Systems), 1 edition (2001).
[Girvan and Newman 2002] Girvan, M. and Newman, M. E. J., “Community structure in social and biological networks”, Proceedings of the National Academy of Sciences, 99, 12, (2002), 7821–7826, doi:10.1073/pnas.122653799.
[Guimerà et al. 2003] Guimerà, R., Danon, L., Díaz-Guilera, A., Giralt, F. and Arenas, A., “Selfsimilar
community structure in a network of human interactions”, Phys. Rev. E, 68, 6, (2003), 065103, doi:10.1103/PhysRevE.68.065103.
[Jay et al. 2009] Jay, G., Hale, J. E., Smith, R. K., Hale, D. P., Kraft, N. A. andWard, C., “Cyclomatic
Complexity and Lines of Code: Empirical Evidence of a Stable Linear Relationship”, JSEA, 2, 3, (2009), 137–143.
[Kennedy et al. 2004] Kennedy, K., Koelbel, C. and Schreiber, R., “Defining and Measuring the Productivity of Programming Languages”, International Journal of High Performance Computing Applications, 18, (2004), 441–448, ISSN 1094-3420, doi: 10.1177/1094342004048537.
[McConnell 2004] McConnell, S., Code Complete: A Practical Handbook of Software Construction, Microsoft Press, Redmond, WA, USA, 2nd edition (2004), ISBN 0735619670.
[Meijer and Gough 2000] Meijer, E. and Gough, J., “Technical Overview of the Common Language Runtime”, Technical Report (2000), http://research.microsoft.com/enus/ um/people/emeijer/Papers/CLR.pdf [accessed 25 March 2012].
[Newman 2001] Newman, M. E. J., “The structure of scientific collaboration networks”, Proceedings of the National Academy of Sciences, 98, 2, (2001), 404–409, doi: 10.1073/pnas.98.2.404.
[Newman 2004] Newman, M. E. J., “Fast algorithm for detecting community structure in networks”, Phys. Rev. E, 69, 6, (2004), 066133, doi:10.1103/PhysRevE.69.066133.
[Newman and Girvan 2004] Newman, M. E. J. and Girvan, M., “Finding and evaluating community structure in networks”, Phys. Rev. E, 69, 2, (2004), 026113, doi: 10.1103/PhysRevE.69.026113.
[Nooy et al. 2004] Nooy,W. d., Mrvar, A. and Batagelj, V., Exploratory Social Network Analysis with Pajek, Cambridge University Press, New York, NY, USA (2004), ISBN 0521602629.
[Nyström et al. 2008] Nyström, J. H., Trinder, P.W. and King, D. J., “High-level distribution for

the rapid production of robust telecoms software: comparing C++ and ERLANG", Concurrency and Computation: Practice and Experience, 20, 8, (2008), 941–968, ISSN 1532-0634, doi:10.1002/cpe.1223.

[Ovelgönne and Geyer-Schulz 2010] Ovelgönne, M. and Geyer-Schulz, A., "Cluster Cores and Modularity Maximization", in W. Fan, W. Hsu, G. I. Webb, B. Liu, C. Zhang, D. Gunopulos and X. Wu (editors), ICDMW '10. 10th IEEE International Conference on Data Mining Workshops (Sydney, Australia), IEEE Computer Society, IEEE Computer Society, Los Alamitos (2010), 1204 – 1213.
[Ovelgönne and Geyer-Schulz 2012] Ovelgönne, M. and Geyer-Schulz, A., "A Comparison of Agglomerative Hierarchical Algorithms for Modularity Clustering", in Proceedings of the 34th Conference of the German Classification Society, Studies in Classification, Data Analysis, and Knowledge Organization, Springer, Heidelberg (2012), 225 – 232.
[Ovelgönne and Geyer-Schulz 2013] Ovelgönne, M. and Geyer-Schulz, A., "An Ensemble Learning Strategy for Graph Clustering", in D. A. Bader, H. Meyerhenke, P. Sanders and D. Wagner (editors), Graph Partitioning and Graph Clustering, Contemporary Mathematics, volume 588, American Mathematical Society, Providence (2013), 187–205.
[Ovelgönne et al. 2010] Ovelgönne, M., Geyer-Schulz, A. and Stein, M., "Randomized Greedy Modularity Optimization for Group Detection in Huge Social Networks", in SNAKDD' 2010: Proceedings of the 4th Workshop on Social Network Mining and Analysis, ACM, New York, NY, USA (2010).
[Paleczny et al. 2001] Paleczny, M., Vick, C. and Click, C., "The Java HotspotTMServer Compiler", in Proceedings of the 2001 Symposium on JavaTM Virtual Machine Research and Technology Symposium - Volume 1, JVM'01, USENIX Association, Berkeley, CA, USA (2001), 1–1.
[Park 1992] Park, R. E., "Software Size Measurement: A Framework for Counting Source Statements",
Technical Report CMU/SEI-92-TR-020, ESC-TR-92-020, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (1992).
[Prechelt 2000] Prechelt, L., "An empirical comparison of seven programming languages", Computer, 33, 10, (2000), 23 –29, ISSN 0018-9162, doi:10.1109/2.876288.
[Prechelt 2011] Prechelt, L., "Plat_Forms: AWeb Development Platform Comparison by an Exploratory
Experiment Searching for Emergent Platform Properties", IEEE Transactions on Software Engineering, 37, 1, (2011), 95–108, ISSN 0098-5589, doi:10.1109/TSE.2010.22.
[Sackman et al. 1968] Sackman, H., Erikson, W. J. and Grant, E. E., "Exploratory experimental studies comparing online and offline programming performance", Commun. ACM, 11, (1968), 3–11, ISSN 0001-0782, doi:10.1145/362851.362858.
[Sjoeberg et al. 2005] Sjoeberg, D. I., Hannay, J. E., Hansen, O., Kampenes, V. B., Karahasanovic,
A., Liborg, N.-K. and Rekdal, A. C., "A Survey of Controlled Experiments in Software Engineering", IEEE Transactions on Software Engineering, 31, 9, (2005), 733–753, ISSN 0098-5589, doi:10.1109/TSE.2005.97.
[Stärk et al. 2012] Stärk, U., Prechelt, L. and Jolevski, I., "Plat_Forms 2011: finding emergent properties of web application development platforms", in Proceedings of the ACMIEEE international symposium on Empirical software engineering and measurement, ESEM '12, ACM, New York, NY, USA (2012), ISBN 978-1-4503-1056-7, 119–128, doi: 10.1145/2372251.2372273.
[Stroustrup 1993] Stroustrup, B., "A History of C++: 1979-1991", in The second ACM SIGPLAN conference on History of programming languages, HOPL-II, ACM, New York, NY, USA (1993), ISBN 0-89791-570-4, 271–297, doi:10.1145/154766.155375.
[Veldhuizen 2003] Veldhuizen, T. L., "C++ Templates are Turing Complete", Technical Report (2003), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.3670 [accessed 10 June 2011].
[Wilbers et al. 2009] Wilbers, I., Langtangen, H. P. and Ødegård, Å., "Using Cython to Speed up Numerical Python Programs", in B. Skallerud and H. I. Andersson (editors), Proceedings of MekIT'09, NTNU, Tapir (2009), ISBN 978-82-519-2421-4, 495–512.

[Zachary 1977] Zachary, W., “An information flow model for conflict and fission in small groups”, Journal of Anthropological Research, 33, (1977), 452–473.