Параллельная обработка больших графов

Занятие 4

А.С. Семенов

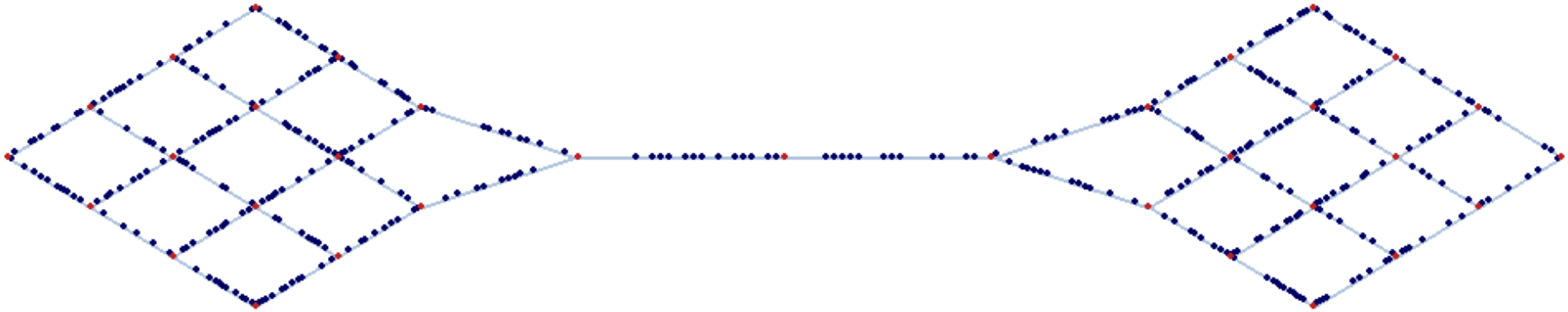
dislab.org

Betweenness centrality

- Связный неориентированный граф
- Допустимы циклы и кратные ребра
- w(e) > 0, e in E
- σ_{st} – количество кратчайших путей между $s, t \in V$
- $\sigma_{st}(v)$ – количество кратчайших путей, проходящих через v

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Betweenness centrality



«Наивный» алгоритм

1. Подсчет количества и длины путей в графе между любыми двумя вершинами
2. Вычисление значения характеристики для каждой вершины

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

«Наивный» алгоритм
1. Подсчет количества путей

Множество предков вершины v на кратчайших путях от s:

$$P_s(v) = \{u \in V \ : \ \{u, v\} \in E, \ d_G(s, v) = d_G(s, u) + \omega(u, v)\}$$

Какое количество путей будет от s до v (S_{sv}) , если известно количество путей от s до u (S_{su})?

«Наивный» алгоритм
1. Подсчет количества путей

$$\sigma_{sv} = \sum_{u \in P_s(v)} \sigma_{su}$$

«Наивный» алгоритм
2. Подсчет характеристики

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Как определить, что кратчайший путь проходит через вершину v?

«Наивный» алгоритм
2. Подсчет характеристики

Критерий Беллмана. Вершина v лежит на кратчайшем пути между вершинами s,t iff

$$d_G(s,t) = d_G(s,v) + d_G(v,t)$$

Как определить количество путей, проходящих через v?

«Наивный» алгоритм
2. Подсчет характеристики

Критерий Беллмана. Вершина v лежит на кратчайшем пути между вершинами s,t iff

$$d_G(s,t) = d_G(s,v) + d_G(v,t)$$

$$\sigma_{st}(v) = \begin{cases} 0 & \text{if } d_G(s,t) < d_G(s,v) + d_G(v,t) \\ \sigma_{sv} \cdot \sigma_{vt} & \text{otherwise} \end{cases}$$

«Наивный» алгоритм

1. Подсчет количества и длины путей в графе между любыми двумя вершинами
2. Вычисление значения характеристики для каждой вершины

Сложность

- Память **$O(N^2)$**
- Время **$O(N^3)$**

Алгоритм U. Brandes (2001)

BC-Brandes (G) // G – связный неориентированный без весов
for all v∈V **do** BC[v] = 0 **end for**
for all s∈V **do**
    BrandesInitS (s)
    ShortestPathsCounting (s)
    DependencyAccumulation ()
end for

Алгоритм Brandes, инициализация

BrandesInitS (s)
initEmptyStack(S) // S - пустой стек
for all t∈V **do**
    initEmptyList(P[t]);
    $s[t] = 0; d[t] = -1; d[v] = 0$
end for
$\sigma[s] = 1; d[s] = 0$

Подсчет числа кратчайших путей

ShortestPathsCounting (s)
initQueue(Q, *s*) // Q – очередь
while Q <> {}
    v = deque(Q); push(S, *v*)
    for all w∈Adj[v]
        if $d[w] < 0$
            enqueue(Q, *w*)
            $d[w] = d[v] + 1$
        end if
        if $d[w] == d[v] + 1$
            $s[w] = s[w] + s[v]$
            append(P[*w*], *v*)
        end if
    end for
end while

Суммирование

$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}} \qquad C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

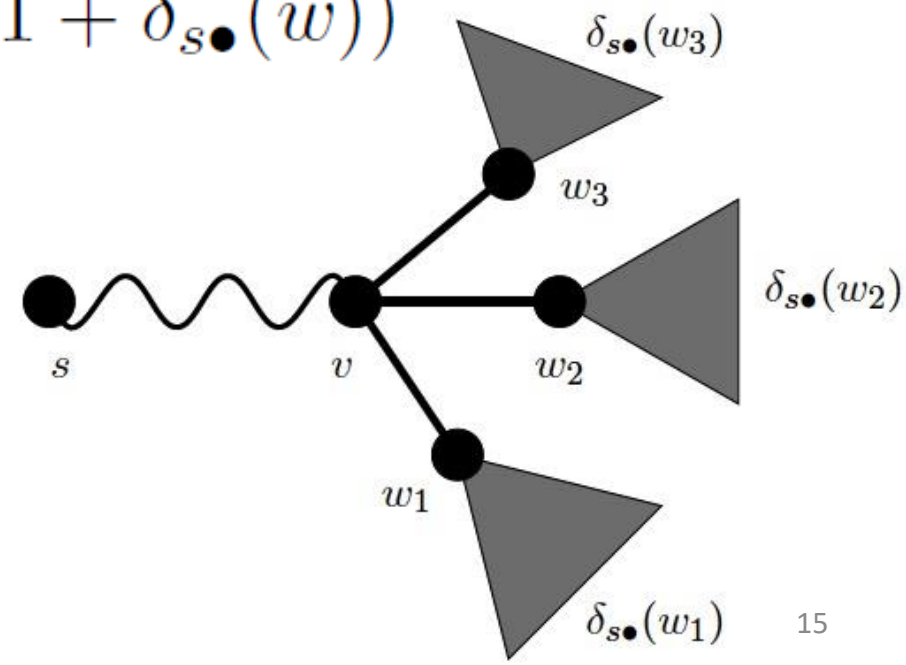
$$C_B(v) = \sum_{s \neq v \neq t \in V} \delta_{st}(v)$$

Суммирование

$$\delta_{s\bullet}(v) = \sum_{t \in V} \delta_{st}(v)$$

Лемма. Если существует только один кратчайший путь из s в t, то

$$\delta_{s\bullet}(v) = \sum_{w\,:\,v \in P_s(w)} (1 + \delta_{s\bullet}(w))$$



Суммирование

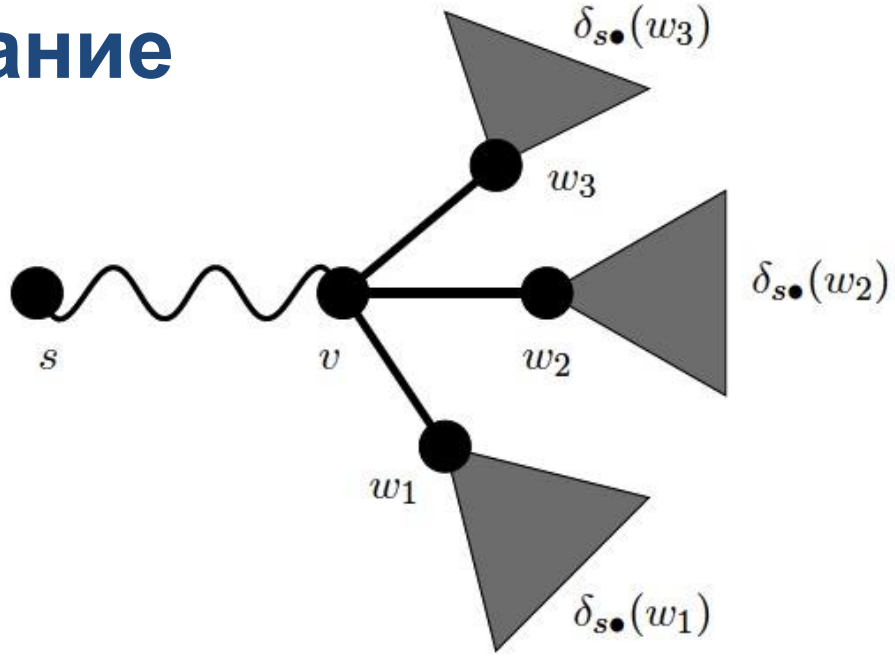
$$W = w : v \in P_s(w)$$

$$\delta_{s\bullet}(v) = \sum_{t \in V} \delta_{st}(v) =$$

$$\sum_{w \in W} \delta_{sw}(v) + \sum_{w \notin W} \delta_{sw}(v) =$$

$$|W| + \sum_{w \in W} \delta_{s\bullet}(w) =$$

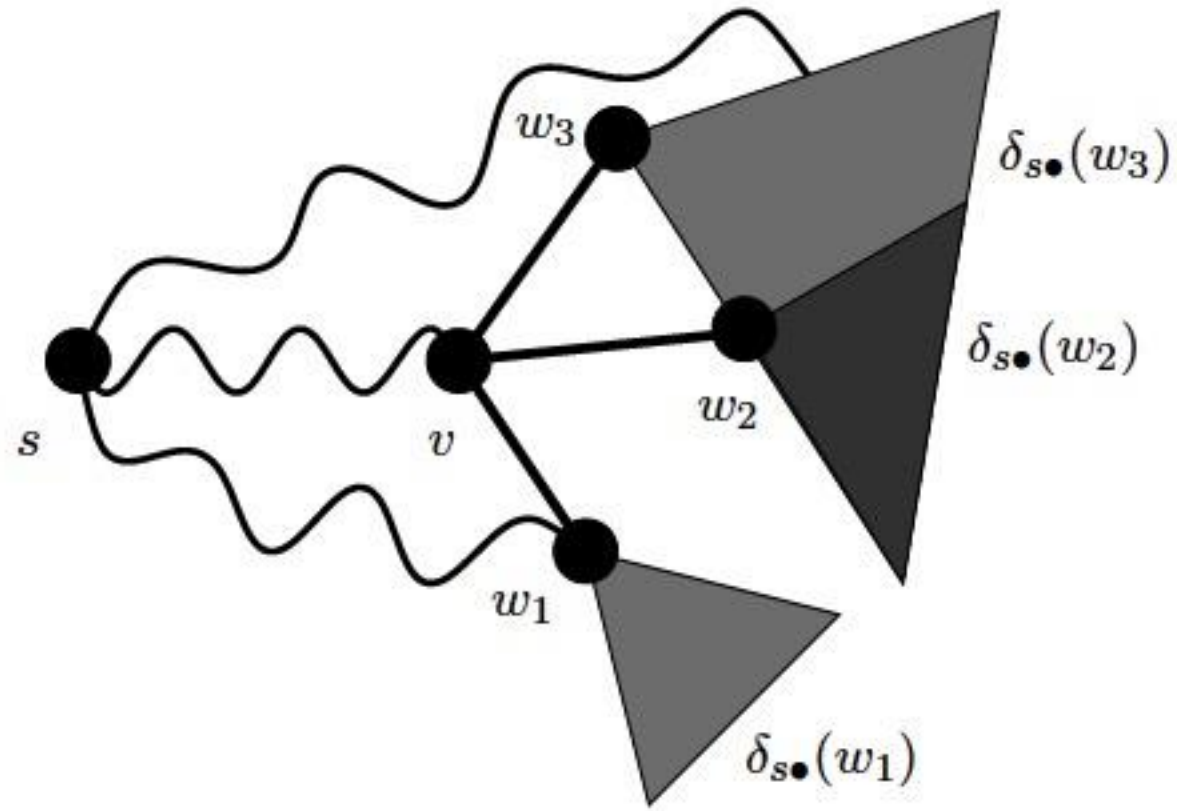
$$\sum_{w \in W} (1 + \delta_{s\bullet}(w))$$



Суммирование

Теорема.

$$\delta_{s\bullet}(v) = \sum_{w\,:\,v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_{s\bullet}(w))$$



Суммирование

DependencyAccumulation ()

while S <> {}

w = pop(S)

for all $v \in P[w]$

$$\delta[v] = \delta[v] + \frac{\sigma[v]}{\sigma[w]} \cdot (1 + \delta[w])$$

end for

if $w <> s$

$$BC[w] = BC[w] + \delta[w]$$

end if

end while

Сложность алгоритма Brandes

```
BC-Brandes (G) // G – связный неориентированный без весов
for all v∈V do BC[v] = 0 end for
for all s∈V do
    BrandesInitS (s)
    ShortestPathsCounting (s)
    DependencyAccumulation ()
end for
```

Сложность:

- Время **O(N M)**
- Память **O(N+M)**

Варианты заданий

Реализовать алгоритм Борувки на распределенной памяти при помощи:

- Функций MPI-1
- mpi4py
- Односторонние коммуникации MPI-3
- Библиотека OpenSHMEM
- Charm++, charm4py
- Chapel
- Apache Spark, GraphX
- Использование ускорителей
- Combinatorial BLAS
- BigData, … ?

- Реализация должна быть масштабируемой **по памяти** и обладать **сильной масштабируемостью**

- **Дедлайны**: 24.04 – корректность
- 15.05 - масштабируемость

MST, алгоритм Борувки

MST-Boruvka (G, w) // G – связный, различные веса
T = { MakeSet(v_0), MakeSet(v_1), …}
A = {}
while |T| > 1
    L = {}
    for all component $C_i \in T$
        (u, v) = argmin W[(u, v)], $u \in C_i$, $v \notin C_i$
            L = L U {(u, v)}
    end for
    for all (u, v) ∈L
        if $u \in C_i$, $v \notin C_i$ **then**
            Union(u, v)
            A = A U {(u, v)}
    end for
end while

Сложность **O(M log N)**

Community Detection

- Задача Community Detection: поиск сообществ в неориентированном графе с весами
- неориентированный граф G = (V, E, W)
- V – множество вершин
- E ⊂ V×V – множество ребер
- W: E → R[0;1] – весовая функция
- Цель задачи — найти такое разбиение графа G на множество C непересекающихся сообществ вершин $c_i \subseteq V$:

$$\cup c_i = V, \forall c_i \in C$$

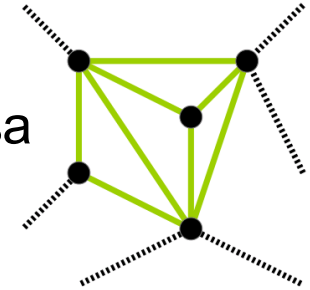
$$c_i \cap c_j = \emptyset, \forall c_i, c_j \in C, i \neq j,$$

Постановка задачи

Необходимо максимизировать функционал модулярности Q:

$$Q = \sum_{c \in C} \left(\frac{\sum_{in}^{c}}{m} - \left[\frac{\sum_{tot}^{c}}{m} \right]^2 \right), \text{ где}$$

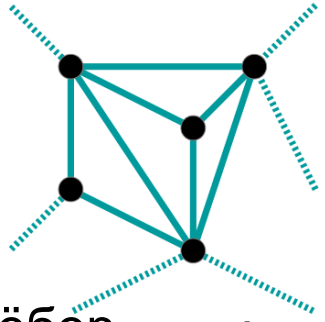
$$\sum_{in}^{c} = \sum_{u,v \in c,\, e(u,v) \in E} w(u,v),$$



$\sum_{in}^{c}$ - Сумма весов всех внутренних рёбер сообщества

$$\sum_{tot}^{c} = \sum_{in}^{c} + \sum_{u \in c, v \notin c, e(u,v) \in E} w(u,v),$$

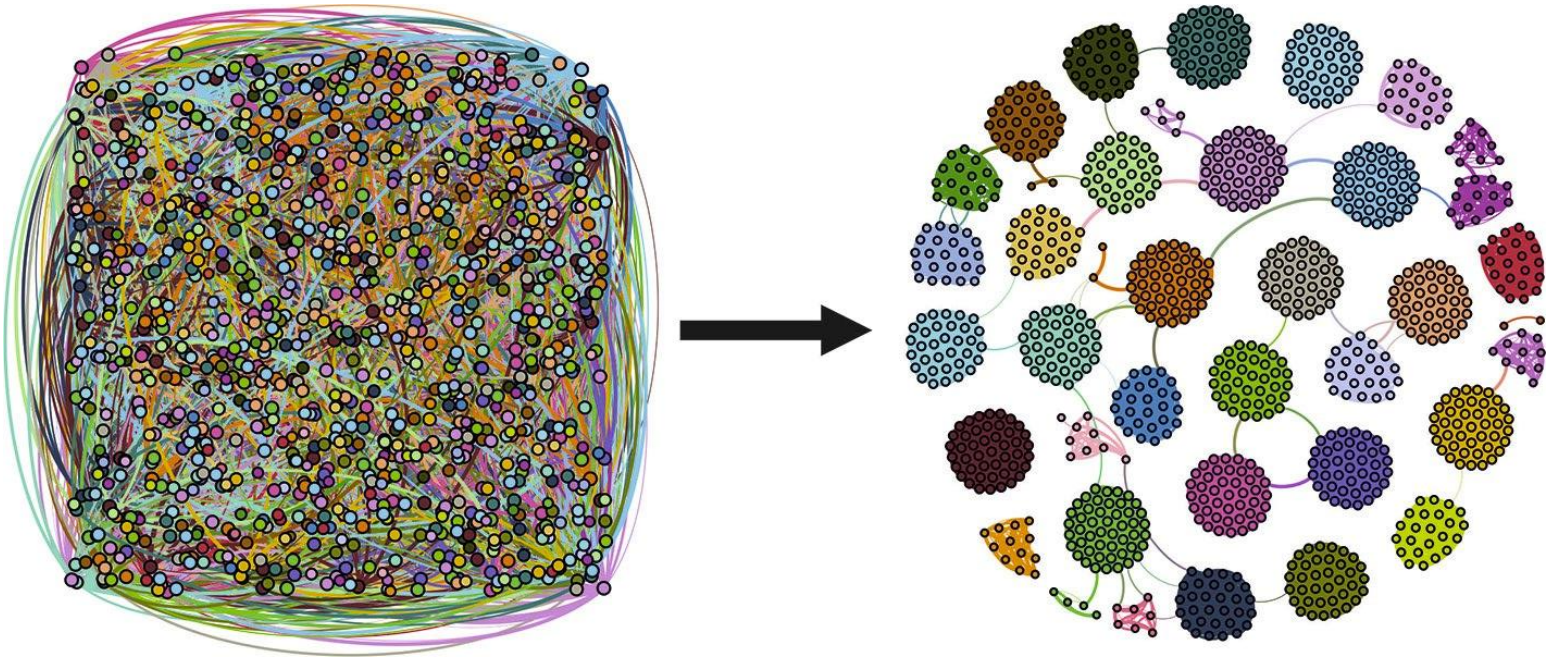
$\sum_{tot}^{c}$ - Сумма весов всех рёбер, исходящих из вершин сообщества



$$m = \sum_{e(u,v) \in E} w(u,v).$$

m - Нормализующий коэффициент, сумма весов всех рёбер графа

Community Detection



SSCA2, N=2^{10}, http://gephi.org

Обзор алгоритмов решения задачи Community Detection

Задача оптимизации модулярности – NP-полная

> 200 алгоритмов

- **Лувенский (Louvain)** $O(m)$
- **Label Propagion** $O(m)$
- **Girvan Newman** $O(nm^2)$
- **Clauset** $O(m\log^2 n)$
- **Рандомизированные алгоритмы**
- **Многоуровневые алгоритмы**

Лувенский алгоритм

- Blondel et al., University of Louvain, 2008
- Локальный жадный алгоритм
- Bottom-up multilevel (восходящий многоуровневый)
- O(m)

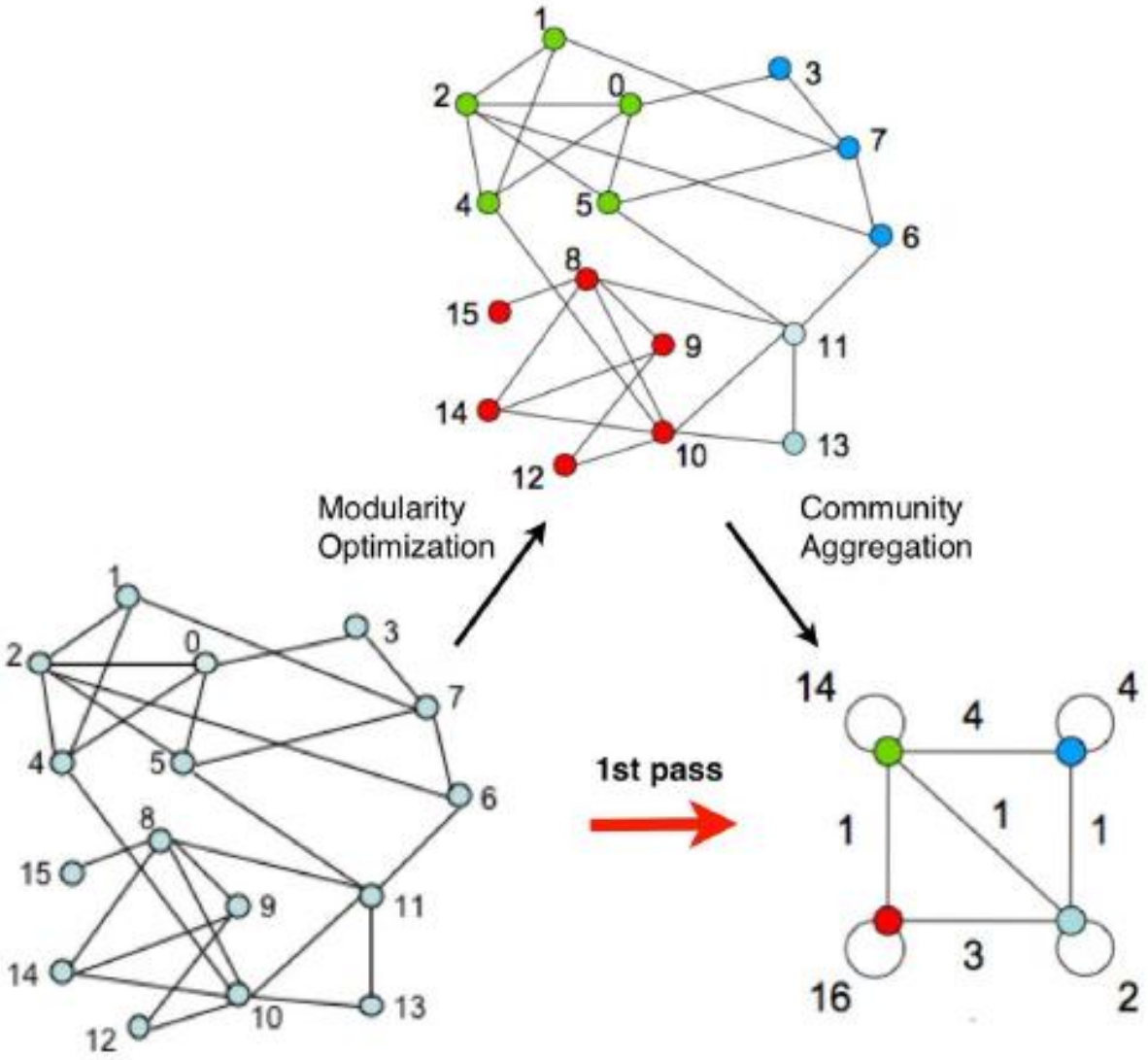
Лувенский алгоритм

```
do
    changed = false
    // Фаза 1. Перемещения
    // Фаза 2. Сжатие
        G(V, E) = compress(G)
while changed
```

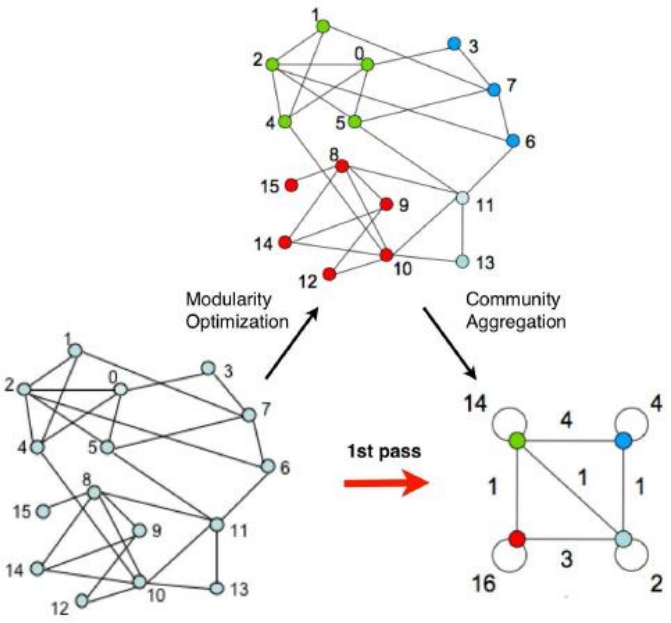
Лувенский алгоритм (2)

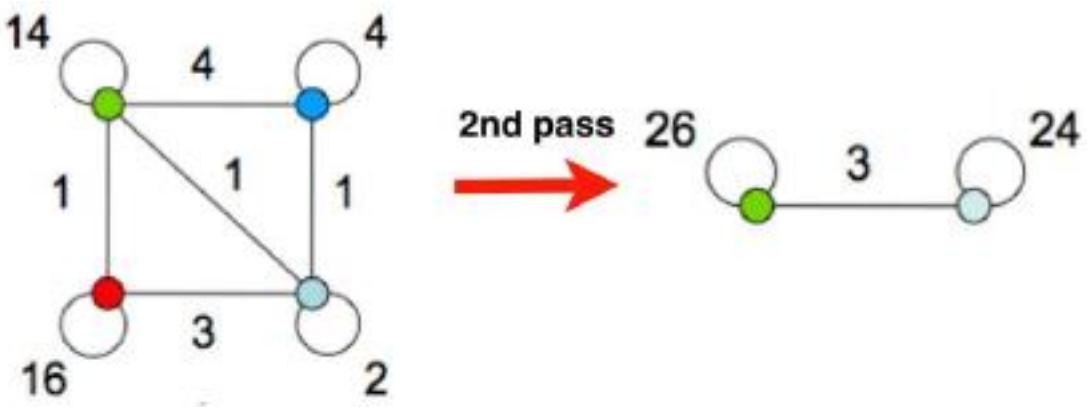
c[u]=u, u∈V
// Фаза 1. Перемещения
do
    more = false
    for all u ∈V
        best_component = argmax_c
                [Δmod(∀c[v], e(u,v)∈E, c[v] ≠ c[u]), Δmod > 0]
        if c[u] ≠ best_component
            update()
            c[u] = best_component
            more = true
        end if
    end for
    changed = changed | more
while more

Лувенский алгоритм (3)



Лувенский алгоритм (4)



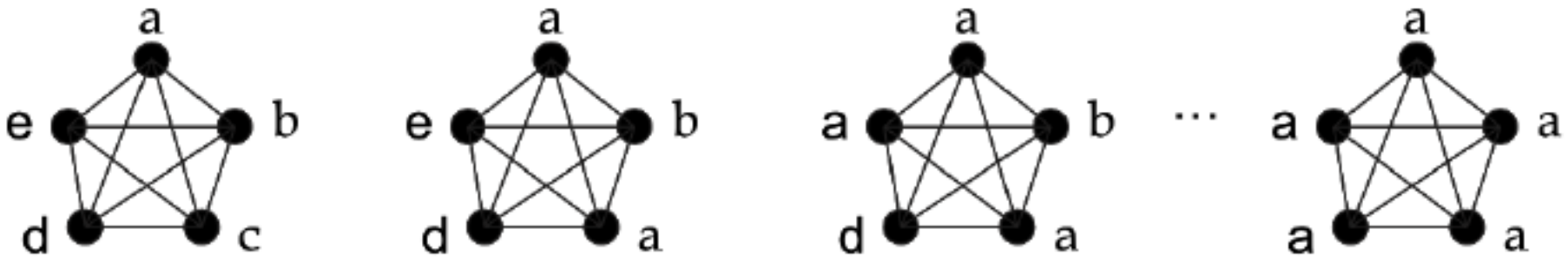


Алгоритм Label Propagation

- Raghavan, USA, 2007
- O(m)
- Количество итераций слабо зависит от размера графа
- Очень прост
- Нет уникального решения
- Иногда может давать плохое решение – одно большое сообщество

Алгоритм Label Propagation

component_id[u]=u, u∈V
do
    updated = false
    for all u ∈V in **random order**
        l = argmax$_c$ [Σ$_c$ w(e), ∀c, e(u,v)∈E, v∈c]
        if component_id[u] ≠ l
            component_id[u] = l
            updated = true
        end if
    end for
while updated



Алгоритм Label Propagation