Система интерактивной визуализации графов

М. А. Крышень

Петрозаводский Государственный Университет
Кафедра Информатики и математического обеспечения

Международная научная конференция
«Дискретная математика, теория графов и их приложения»
11–14 ноября 2013 г.

Задача

- Визуализация графа, содержащего порядка тысяч вершин.
- Граф строится на основе исходных данных в соответствии с набором правил.
- Вершины — сложные интерактивные объекты:

  взаимодействие с вершинами может приводить к изменению графа.
- Обновление раскладки при изменении графа.
- Приемлемая производительность и эффективное использование вычислительных ресурсов:

  распараллеливание работы на несколько процессорных ядер.

Основные области применения:

- исследование работы вычислительных сетей,
- сетевое управление.

Метод визуализации

Исходный граф

Данные предметной области

Правила преобразования

Предметно-ориентированный язык на основе Clojure (Лисп).

Предоставляется библиотека комбинируемых визуальных объектов.

Граф интерактивных визуальных объектов

Алгоритм раскладки графа

Визуализация графа методом физических аналогий.

Интерактивное визуальное представление графа

Изменение правил визуализации и обновление графа в ответ на действия пользователя.

Абстрактный тип данных графа

Coll[E] — коллекция элементов типа E.

Context[V], V — тип вершин

- vertex : Context[V] $\to V$
- present : Context[V] $\to \{0, 1\}$
- incoming : Context[V] $\to$ Coll[Context[V]]
- outgoing : Context[V] $\to$ Coll[Context[V]]

Graph

- contexts : Graph[V] $\to$ Coll[Context[V]]
- context : Graph[V] $\times V \to$ Context[V]

Построение графа

Неизменяемые структуры данных:

- порождение новой версии вместо изменения,
- новая версия разделяет память со старой.

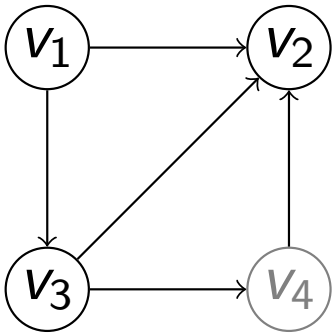
PersistentGraph$[V]$ <: Graph$[V]$

- into : PersistentGraph$[V]$ $\times$ Graph$[V]$ $\to$ PersistentGraph$[V]$
- remove : PersistentGraph$[V]$ $\times$ Graph$[V]$ $\to$ PersistentGraph$[V]$

Создание графов типа звезда:

- star : $V \times$ Coll$[V] \times$ Coll$[V] \times \{0, 1\} \to$ *Graph*$[V]$

Структура данных графа



Persistent hash-map и vector — неизменяемые структуры данных, предоставляемые Clojure.

			:vertex	:presence	:in	:out
v_1	0	0	(v_1,	1,	$\varnothing$,	$\{1, 3\}$)
v_2	3	1	(v_3,	1,	$\{0\}$,	$\{3, 4\}$)
v_3	1	2	(v_4,	0,	$\{1\}$,	$\{3\}$)
v_4	2	3	(v_2,	1,	$\{0, 1, 2\}$,	$\varnothing$)

persistent hash-map

persistent vector

Правила преобразования

- Предметно-ориентированный язык.
- Произвольная фильтрация и агрегация вершин.
- S — тип вершин исходного графа,
  View — интерактивный визуальный объект.
  Преобразование Graph[S] (модель предметной области)
  в Graph[View].
  Правило — пара функций: p, f,
  - $p : S \to \mathrm{Coll}[S]$
    для вершины исходного графа возвращает список вершин, которые должны быть объединены,
  - $f : \mathrm{Coll}[S] \to \mathrm{View}$
    для списка вершин возвращает интерактивный визуальный объект.

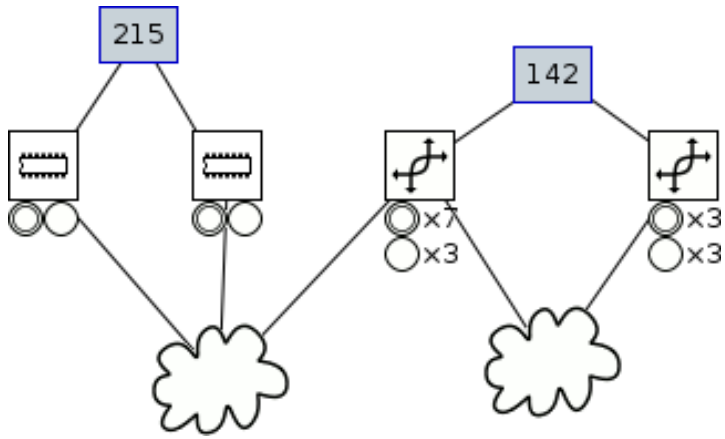
  Правило применимо, если p возвращает непустую коллекцию.
- Clojure (диалект Лиспа).

Пример правил преобразования

```
;; Комната и все ее отделы
(path Room Occupancy)
  ;; Номер комнаты в прямоугольнике
  #(-> % first :number label (panel 5) border)

;; Устройство и все его канальные и сетевые интерфейсы
(path Device LinkInterface NetworkInterface)
  ;; Набор элементов, ограничение координаты y=0
  #(-> % stack (const-y 0))
```

Network stack



Абстрактный тип данных раскладки

Point — тип данных, представляющий точки пространства,
Interval — прямоугольник.

LayoutContext$[V]$ <: Context$[V]$

- location : LayoutContext$[V] \to$ Point
- bounds : LayoutContext$[V] \to$ Interval

Layout$[V]$ <: Graph$[V]$

- layout-bounds : Layout$[V] \to$ Interval
- search : Layout$[V] \times$ Interval $\to$ Coll[LayoutContext$[V]$]
- paths : Layout$[V] \times$ Interval $\to$
  (LayoutContext$[V]$, LayoutContext$[V]$, Coll[Point])

Абстрактный тип данных раскладки (продолжение)

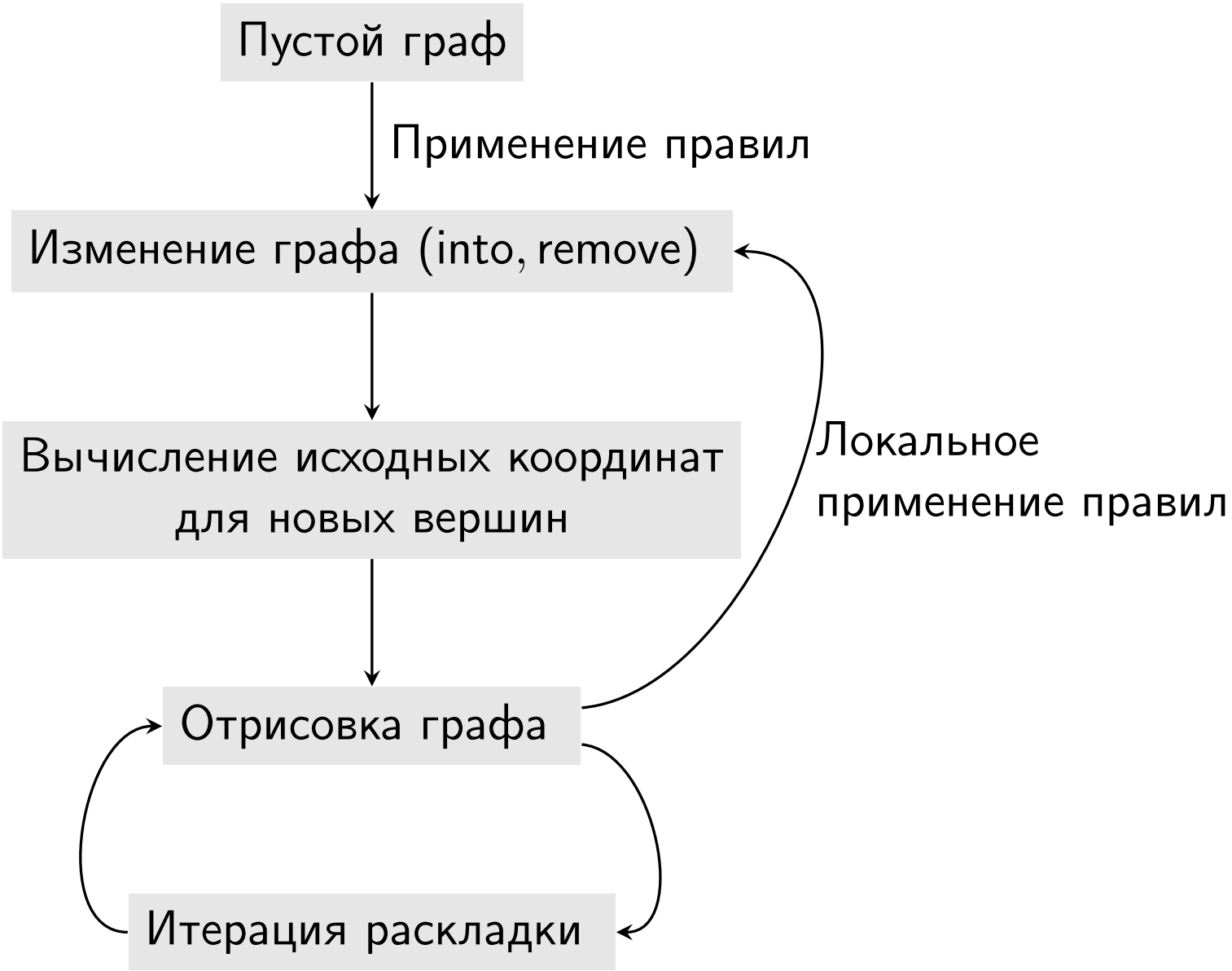
IterativeLayout$[V]$ <: Layout$[V]$

- update : IterativeLayout$[V]$ $\to$ IterativeLayout$[V]$
- complete? : IterativeLayout$[V]$ $\to \{0, 1\}$

Constraints$[V]$

- preferred-bounds : Constraints$[V]$ $\times$ Context$[V]$ $\to$ Interval
- anchor : Constraints$[V]$ $\times$ Context$[V]$ $\times$ Context$[V]$ $\to$ Point
- constraint-location : Constraints$[V]$ $\times$ Context$[V]$ $\times$ Point $\to$ Point

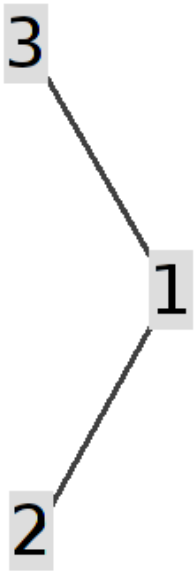
Вычисление раскладки



Вычисление исходных координат

- Минимальное время отклика при изменениях.
  Рассматриваются только новые и смежные с новыми вершины.
- Предсказуемый результат при небольших локальных изменениях.

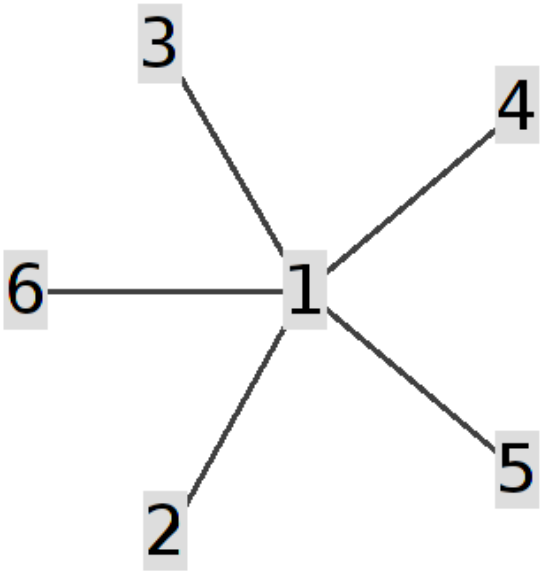
Обход нового подграфа в ширину и определение координат вершин для каждого уровня:



Вычисление исходных координат

- Минимальное время отклика при изменениях.
  Рассматриваются только новые и смежные с новыми вершины.
- Предсказуемый результат при небольших локальных изменениях.

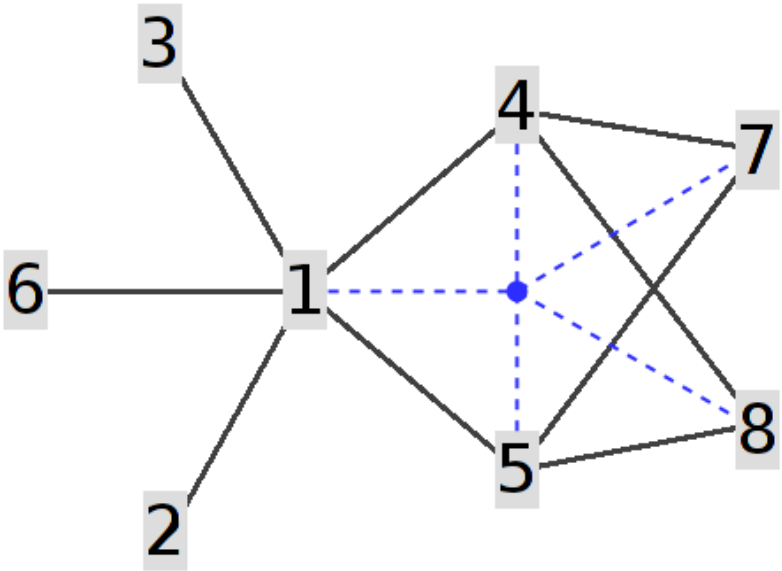
Обход нового подграфа в ширину и определение координат вершин для каждого уровня:



Вычисление исходных координат

- Минимальное время отклика при изменениях.
  Рассматриваются только новые и смежные с новыми вершины.
- Предсказуемый результат при небольших локальных изменениях.

Обход нового подграфа в ширину и определение координат вершин для каждого уровня:



Раскладка на основе физических аналогий

Система заряженных частиц — вершин и пружин — ребер.

Физическая модель: сила притяжения: $f_a(d) = c_a d$, c_a — константа;
отталкивания: $f_r(d) = c_r/d^2$, c_r — константа.
Eades, 1984: $f_a(d) = c_a \log(d)$, $f_r(d) = -c_r/d^2$
Fruchterman & Reingold, 1991: $f_a(d) = d^2/k$, $f_r(d) = -k^2/d$,
k — оптимальное расстояние.

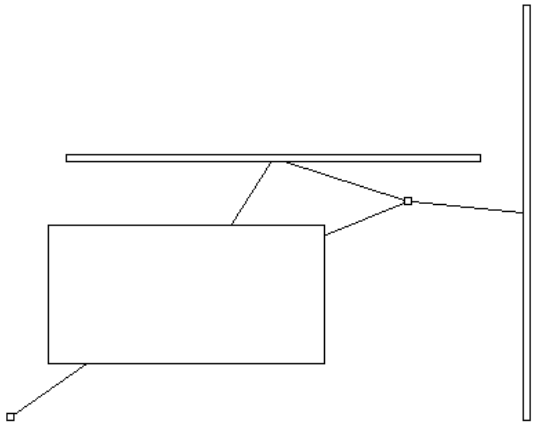
Итерация алгоритма (Fruchterman & Reingold):

1. Вычисление сил, действующих на каждую вершину v:
$$F_v = \sum_{u \in V} f_r(|\Delta_{u,v}|)\frac{\Delta_{u,v}}{|\Delta_{u,v}|} + \sum_{u \in V : \{u,v\} \in E} f_a(|\Delta_{u,v}|)\frac{\Delta_{u,v}}{|\Delta_{u,v}|},$$
V, E — множества вершин и ребер, $\Delta_{u,v}$ — вектор расстояния.
2. Перемещение вершин (ограничено параметром — температурой).
3. Уменьшение температуры и проверка завершения.

Минимизация энергии $\sum_{v \in V} F_v^2$.

Модификации алгоритма раскладки

- Учет размеров вершин: вычисление сил относительно ограничивающих прямоугольников.



- Ограничение сил отталкивания на ранних итерациях.
- k-d дерево для группировки слабо взаимодействующих зарядов-вершин: сложность итерации $O(n \log n)$ вместо $O(n^2)$.
- Адаптивная температура (Hu, 2005):
  - $t_i = ct_{i-1}$, $c = c_h > 1$, если несколько итераций подряд энергия уменьшалась, иначе $c = c_d, 0 < c_d < 1$.
  - Корректировка раскладки после изменений начинается с небольшой температуры.
- Параллельная реализация.

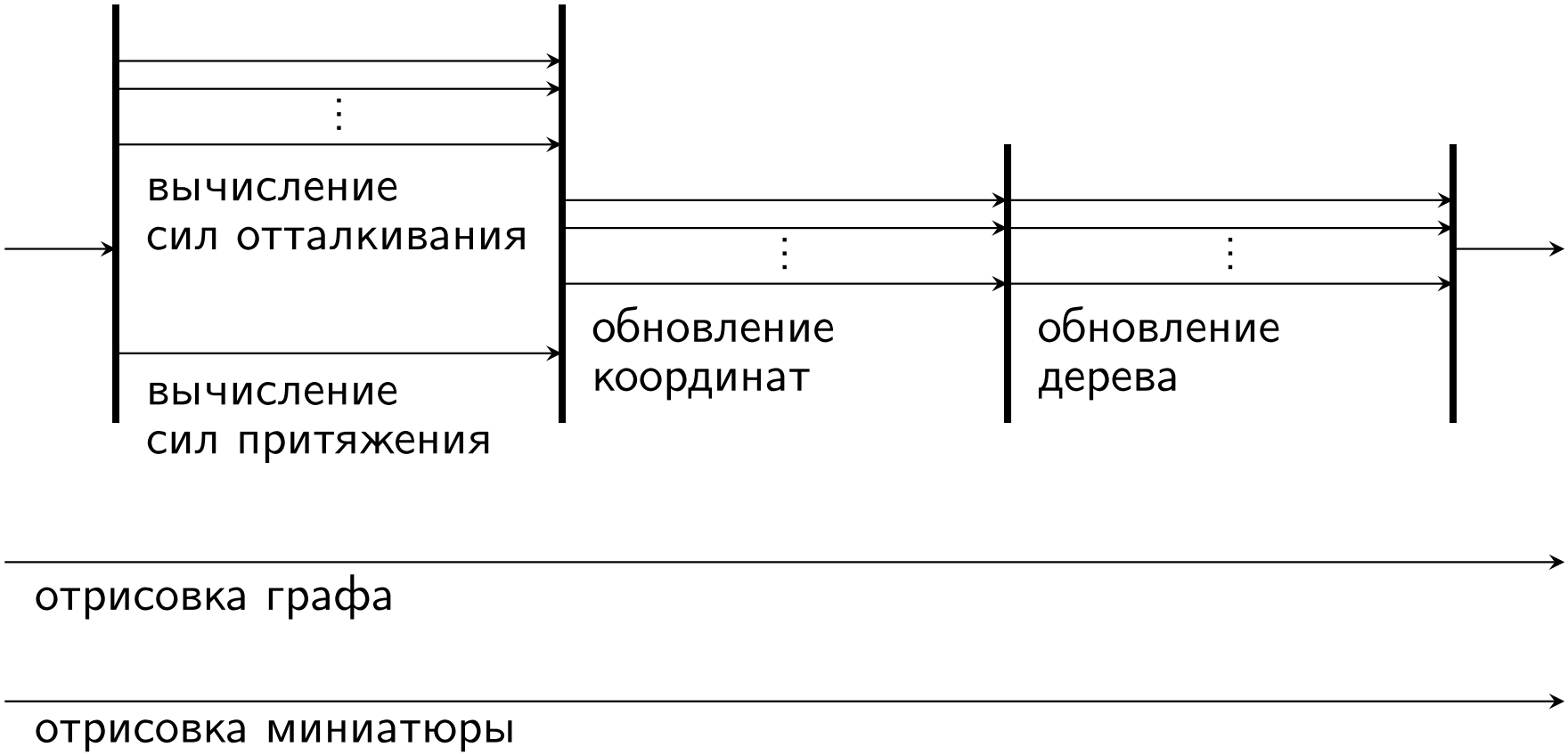
Алгоритм раскладки графа: обход k-d дерева

Для каждой вершины графа.

n — текущий узел дерева, изначально — корень,
d_n — наибольшая сторона узла n,
r_n — расстояние от центра масс узла до границы вершины графа,
c — константа.

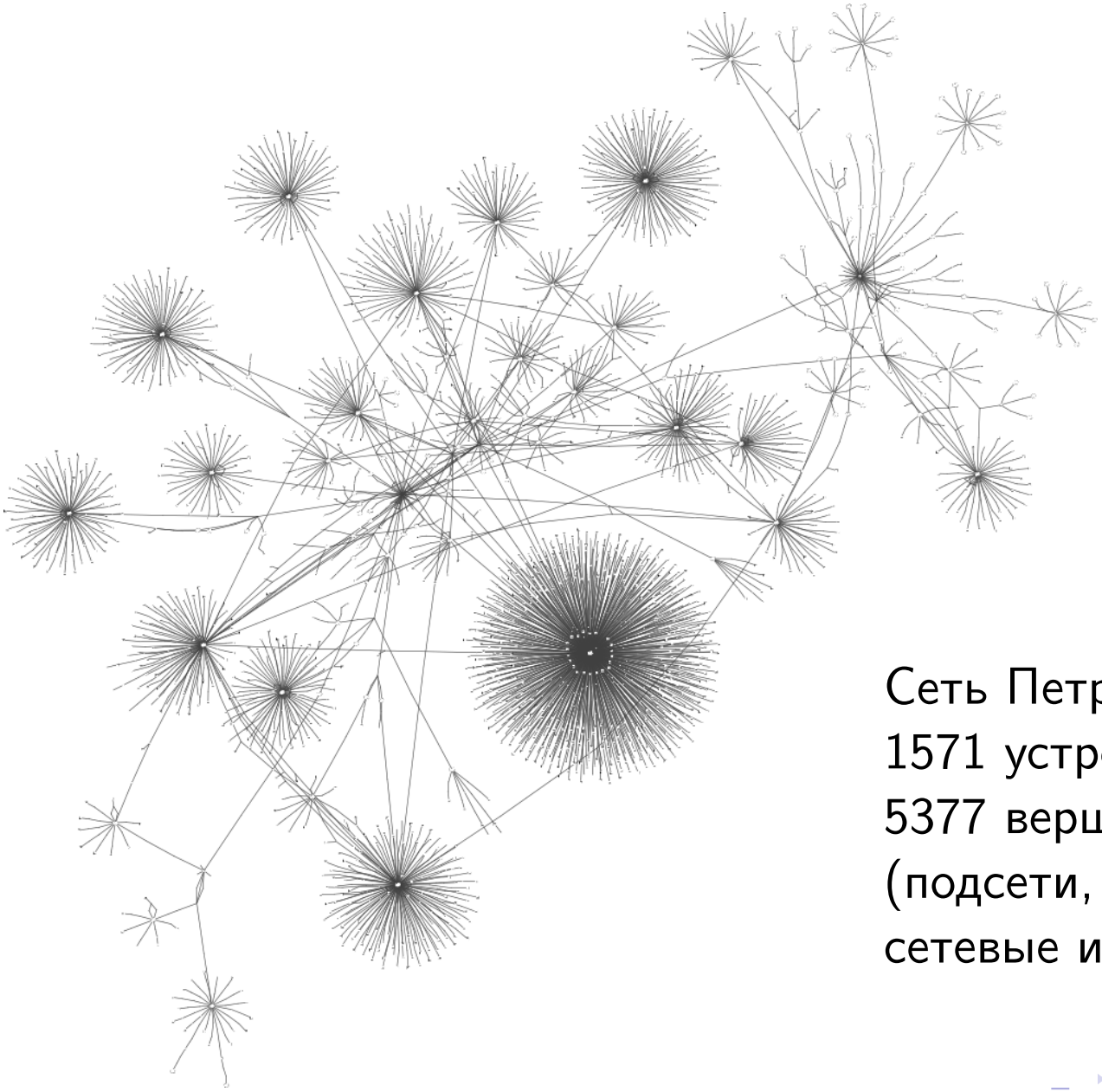
1. Если узел n не листовой и $d_n/r_n > c$, выполняем алгоритм для каждого потомка узла n.
2. Иначе вычисляем силу отталкивания для вершины графа относительно центра масс узла n.

Параллельная реализация



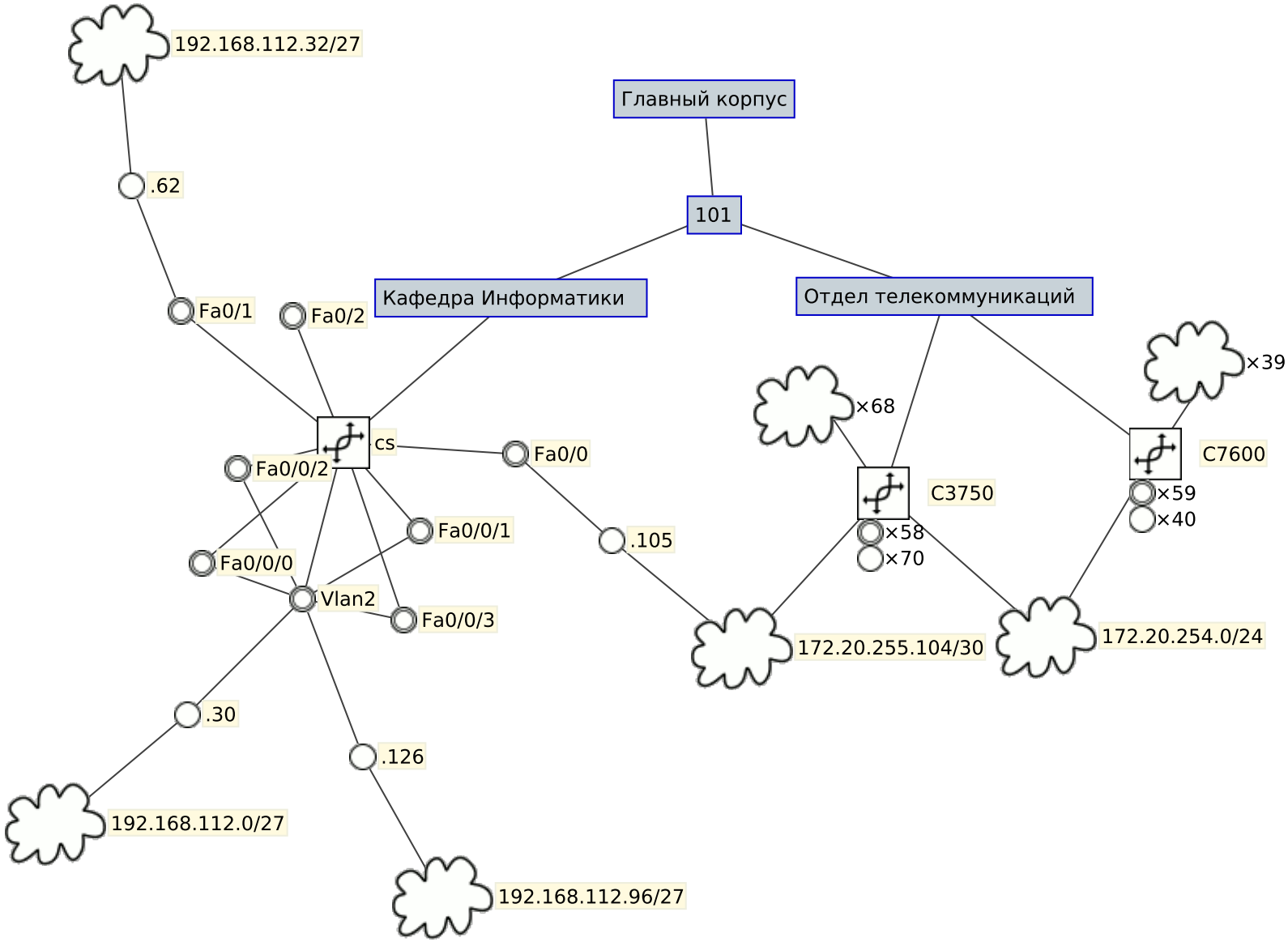
Визуализация графа вычислительной сети ПетрГУ (5377 вершин) занимает 11,5 с на ЦП Intel i5-3330 (360 итераций по 32 мс).

Пример

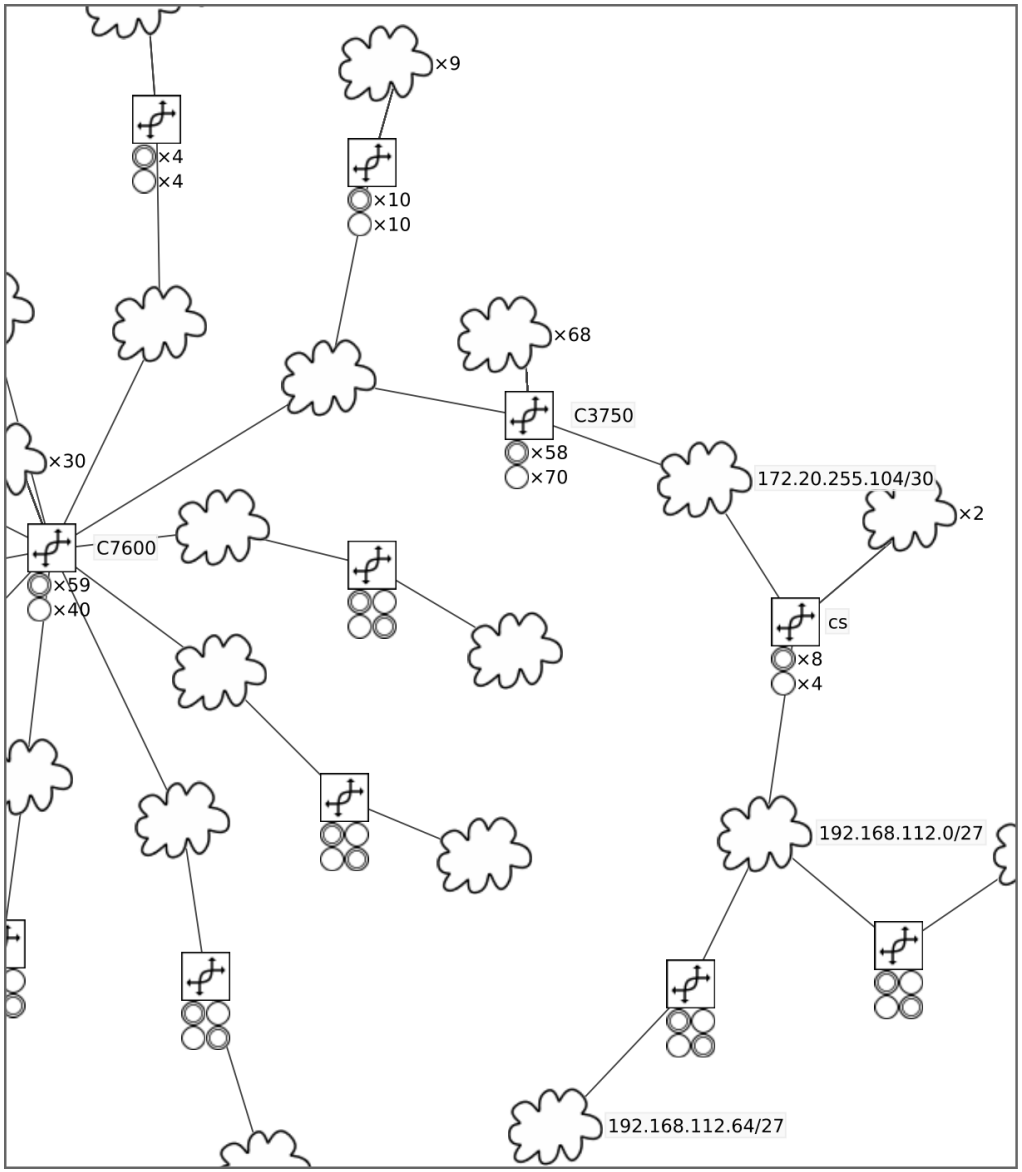


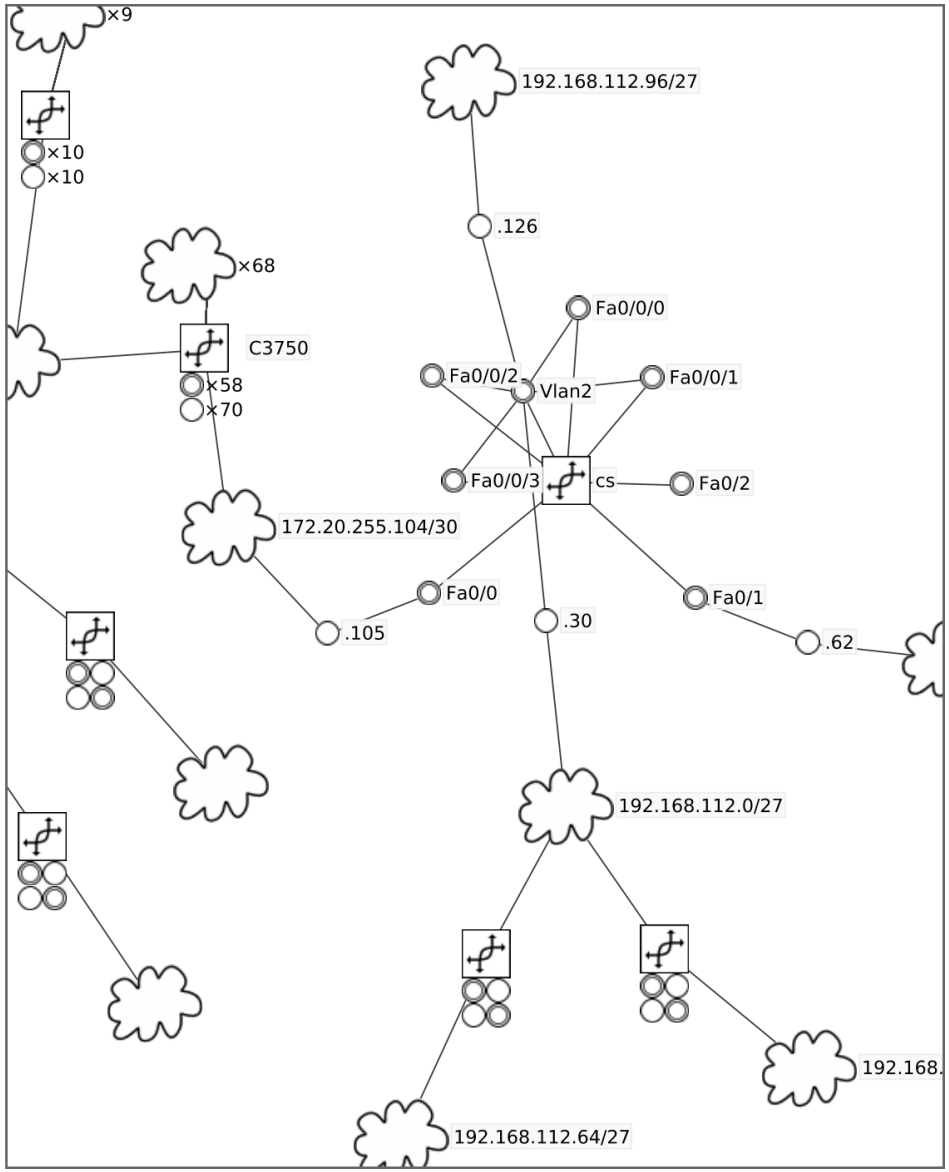
Сеть ПетрГУ:
1571 устройство,
5377 вершин в графе
(подсети, устройства и
сетевые интерфейсы).

Пример


192.168.112.32/27
.62
Fa0/1
Fa0/2
Главный корпус
101
Кафедра Информатики
Отдел телекоммуникаций
cs
Fa0/0/2
Fa0/0
Fa0/0/1
Fa0/0/0
Vlan2
Fa0/0/3
.105
.30
.126
192.168.112.0/27
192.168.112.96/27
172.20.255.104/30
×68
C3750
×58
×70
×39
C7600
×59
×40
172.20.254.0/24

Пример


×9
×4
×4
×10
×10
×68
C3750
×58
×70
×30
C7600
×59
×40
172.20.255.104/30
×2
cs
×8
×4
192.168.112.0/27
192.168.112.64/27


×9
×10
×10
192.168.112.96/27
.126
×68
C3750
×58
×70
Fa0/0/0
Fa0/0/2
Vlan2
Fa0/0/1
Fa0/0/3
cs
Fa0/2
172.20.255.104/30
Fa0/0
Fa0/1
.105
.30
.62
192.168.112.0/27
192.168.112.64/27
192.168.

Спасибо за внимание.