Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ ПО ПРАКТИКУМУ №2
по курсу «Архитектура ЭВМ»

«Обработка и визуализация графов
в вычислительном комплексе Тераграф»

Студент Золотухин Алексей Вячеславович

Группа ИУ7-54Б

Оценка (баллы) _____

Преподаватель Ибрагимов С.В.

2022 г.

Содержание

Введение

Практикум посвящен освоению принципов представления графов и их обработке с помощью вычислительного комплекса Тераграф. В ходе практикума необходимо ознакомиться с вариантами представления графов в виде объединения структур языка C/C++, изучить и применить на практике примеры решения некоторых задач на графах.

По индивидуальному варианту необходимо разработать программу хост-подсистемы и программного ядра sw_kernel, выполняющего обработку и визуализацию графов.

1 Теоретические сведения о визуализации графов

Визуализация графа — это графическое представление вершин и ребер графа. Визуализация строится на основе исходного графа, но направлена на получение дополнительных атрибутов вершин и ребер: размера, цвета, координат вершин, толщины и геометрии ребер. Помимо этого, в задачи визуализации входит определение масштаба представления визуализации. Для различных по своей природе графов, могут быть более применимы различные варианты визуализации. Таким образом задачи, входящие в последовательность подготовки графа к визуализации, формулируются исходя из эстетических и эвристических критериев.

1.1 Алгоритмы раскладки графов

Раскладка неориентированных графов испольуется при проектировании топологии СБИС, целью которого является оптимизация схемы для получения наименьшего количества пересечений линий.

Пружинная система запускается со случайным начальным состоянием, и вершины соответственно перемещаются под действием пружинных сил. Оптимальная компоновка достигается за счет того, что энергия системы сводится к минимуму.

1.1.1 The Spring Model

Модель spring-embedder была первоначально предложена Eades (1984) и в настоящее время является одним из самых популярных алгоритмов для рисования неориентированных графов с прямолинейными ребрами, широко используемого в системах визуализации информации.

Алгоритм Идеса следует двум эстетическим критериям: равномерная длина ребер и максимально возможная симметрия. В модели Spring-Embedder вершины графа обозначаются набором колец, и каждая пара

колец соединена пружиной. Пружина связана с двумя видами сил: силами притяжения и силами отталкивания, в зависимости от расстояния и свойств соединительного пространства.

1.1.2 Local Minimum

Модель spring-embedder привела к созданию ряда модифицированных и расширенных алгоритмов раскладки неориентированных графов. Например, силы отталкивания обычно вычисляются между всеми парами вершин, а силы притяжения могут быть рассчитаны только между соседними вершинами. Упрощенная модель уменьшает временную сложность: вычисление сил притяжения между соседями выполняется за O(|E|), хотя вычисление силы отталкивания выполняется за O($|V|^2$), что в является недостатком алгоритмов с n телами. Камада и Каваи (1989) представили алгоритм, основанный на модели пружинного внедрения Идса, который пытается достичь следующих двух критериев или эвристик рисования графа: количество пересечений ребер должно быть минимальным, а вершины и ребра распределены равномерно.

Цель алгоритма состоит в том, чтобы найти локальный минимум энергии в соответствии с вектором градиента $\sigma = 0$, что является необходимым, но не достаточным условием глобального минимума энергии. С точки зрения вычислительной сложности, такой поиск требует большого количества операций, поэтому в реализацию часто включаются дополнительные элементы управления, чтобы гарантировать, что пружинная система не окажется в локальном минимуме. В отличие от алгоритма Идеса, который явно не включает закон Гука, алгоритм Камады и Каваи перемещает вершины в новые положения по одной, так что общая энергия пружинной системы уменьшается с новой конфигурацией. Он также вводит понятие желаемого расстояния между вершинами на визуализации: расстояние между двумя вершинами пропорционально длине кратчайшего пути между ними.

1.1.3 Force-Directed Placement

Алгоритм Фрухтермана-Рейнгольда основан на модели пружинного встраивания Идса. Он равномерно распределяет узлы, минимизируя пересечения ребер, а также поддерживает одинаковую длину ребер. В отличие от алгоритма Камада-Каваи, алгоритм Фрухтермана-Рейнгольда использует две силы (силы притяжения и силы отталкивания) для обновления узлов, а не использует функцию энергии с теоретическим графическим расстоянием.

Алгоритм Фрухтермана-Рейнгольда выполняется итеративно, и все узлы перемещаются одновременно после расчета сил для каждой итерации. Алгоритм добавляет атрибут «смещения» для контроля смещения положения узлов. В начале итерации алгоритм Фрухтермана-Рейнгольда вычисляет начальное значение смещения для всех узлов с использованием силы отталкивания (fr). Алгоритм также использует силу притяжения (fa) для многократного обновления визуального положения узлов на каждом ребре. Наконец, он обновляет смещение положения узлов, используя значение смещения.

2 Выполнение практикума

Мой вариант №6, поэтому визуализировались данные из необходимого файла №6.

Для визуализации графа был использован код примера с модификацией в файле исходных текстов было реализовано простое чтение текстовых файлов для удобной загрузке большого количества вершин и связей между ними напрямую из файла;

Листинг 2.1 – Модификация файла host_main.cpp

```
1  unsigned int*
       host2gpc_ext_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
2  unsigned int messages_count = 0;
3  unsigned int u, v, w;
4
5  __foreach_core(group, core)
6  {
7      host2gpc_ext_buffer[group][core] = (unsigned
           int*)lnh_inst.gpc[group][core]->external_memory_create_buffer(16
           * 1048576 * sizeof(int)); //2*3*sizeof(int)*edge_count);
8      offs = 0;
9
10     std::ifstream file(argv[3], std::ios::in);
11
12     if (!file.is_open())
13     {
14         std::cout << "Cannot open input file." << std::endl;
15
16         return EXIT_FAILURE;
17     }
18
19     for (std::string line; std::getline(file, line); )
20     {
21         std::vector<std::string> tokens = split(line, '\t');
22
23         if (tokens.size() != 2)
24         {
25             for (auto token : tokens)
26             std::cout << token << " ";
27             std::cout << std::endl;
28             std::cout << "Incorrect tokens count in file: "
```

```
                expected␣3,␣got␣" << tokens.size() << "." <<
                std::endl;
29
30              return EXIT_FAILURE;
31          }
32
33          u = std::stoul(tokens[0]);
34          v = std::stoul(tokens[1]);
35          w = 1;
36
37          //Граф долженбытьсвязным
38          EDGE(u, v, w);
39          EDGE(v, u, w);
40          messages_count += 2;
41      }
42
43      lnh_inst.gpc[group][core]->external_memory_sync_to_device(0,
            3 * sizeof(unsigned int) * messages_count);
44  }
```

Полученные изображения графа в box-раскладке и force-раскладке приведены на рисунках 2.1 – 2.2.

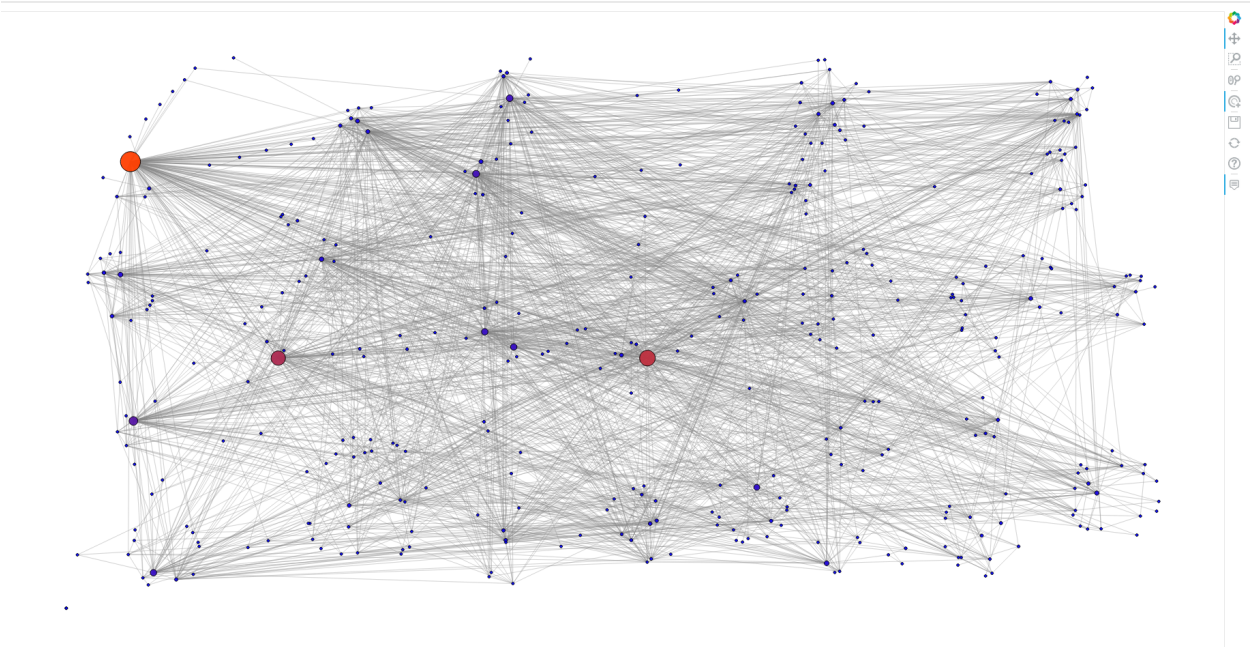

Рисунок 2.1

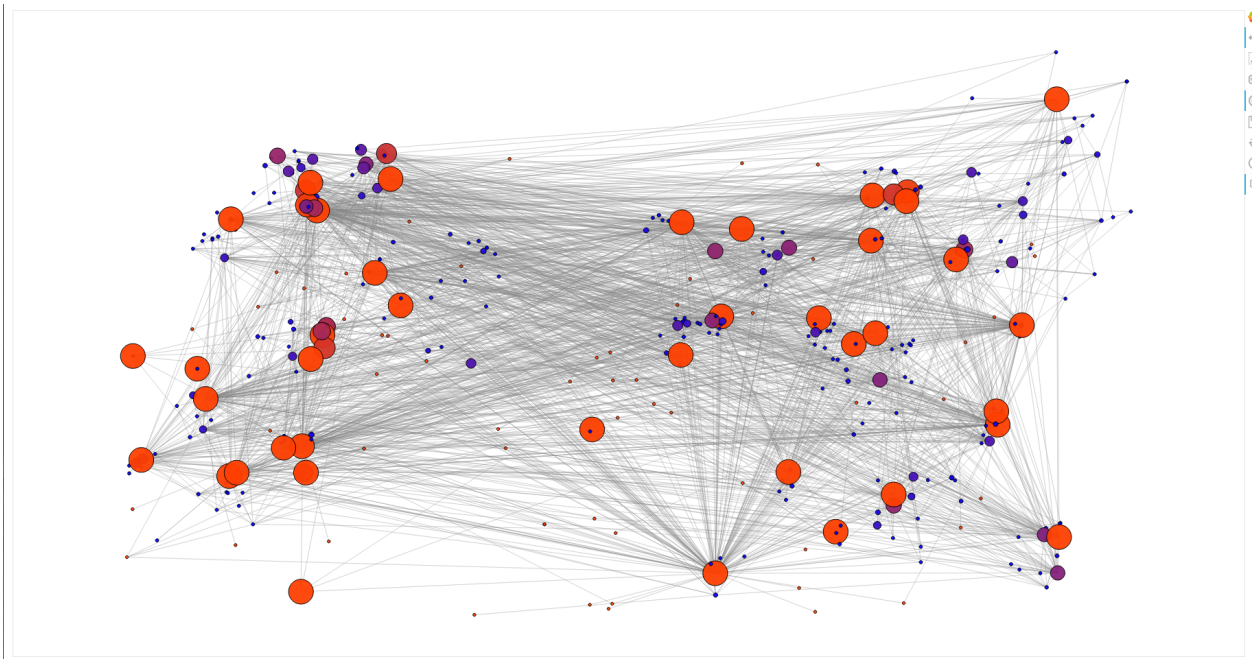

Рисунок 2.2

Заключение

В данном практикуме были рассмотрены принципы визуализации графов, рассмотрены различные алгоритмы визуализации графов и раскладки графов. Было выполнено задание согласно индивидуальному варианту.