ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

УДК 519.173.2

РАВНОМЕРНАЯ ПОУРОВНЕВАЯ УКЛАДКА ГРАФОВ

Карлов Б.Н., Наймушин А.В.
Тверской государственный университет, г. Тверь

Поступила в редакцию 28.05.2018, после переработки 22.06.2018.

В статье рассматривается алгоритм поуровневой укладки ациклических ориентированных графов. Предложен метод для распределения вершин графа по уровням, при котором вершины пути укладываются на уровни с приблизительно равным шагом. Описанный алгоритм сначала распределяет по уровням вершины, лежащие на самых длинных путях графа. После этого алгоритм укладывает на эти же уровни оставшиеся вершины, при этом еще не уложенные пути перебираются по убыванию длины. Для нахождения еще не уложенных длинных путей используется модифицированный метод поиска путей в ациклических графах, основанный на поиске в глубину и топологической сортировке. Доказано, что временная сложность описанного алгоритма при работе на графе $G = (V, E)$ составляет $O(|V||E|)$. Были проведены вычислительные эксперименты, которые показали, что для графов не очень большого размера с относительно маленькой плотностью время работы алгоритма является приемлемым для практического использования.

Ключевые слова: укладка графа на плоскость, распределение вершин по уровням, алгоритм Сугиямы, временная сложность.

Вестник ТвГУ. Серия: Прикладная математика. 2018. № 2. С. 85–98.
https://doi.org/10.26456/vtpmk496

Введение

Задача визуализации графов имеет многочисленные практические применения в различных областях информатики, таких как разработка компиляторов, базы данных, разработка программного обеспечения и графических интерфейсов и т.п. К другим приложениям относятся анализ графических данных, в том числе в общественных науках, и вообще визуализация различных типов схем и диаграмм. На практике часто встречаются различные иерархические структуры, такие как PERT-диаграммы, диаграммы наследования классов в объектно-ориентированных языках программирования и т.п. Стандартный подход к визуализации таких графов состоит в том, что вершины графа распределяются по нескольким горизонтальным уровням, так чтобы все ребра были направлены сверху вниз. Первые алгоритмы такого типа были описаны в [4] и [10]. Наиболее распространенный метод был предложен в [9]. Укладка графа проводится в четыре

этапа. На первом этапе граф преобразуется в ациклический. На втором этапе вершины полученного ациклического графа распределяются по уровням. На третьем этапе вершины на каждом уровне упорядочиваются так, чтобы минимизировать число пересечений ребер. Наконец, на четвертом этапе вычисляются координаты вершин. Эффективная версия этого алгоритма приведена в [6].

В настоящей статье мы рассматриваем задачу распределения вершин по уровням. Для этой задачи было разработано множество подходов. Некоторые из них описаны в [3, 8]. Один из простейших алгоритмов помещает каждую вершину на уровень с номером равным ее глубине. Он укладывает вершины на минимально возможное число уровней и допускает реализацию за линейное время с использованием топологической сортировки (см. [1]). Недостатком этого метода является то, что он может получить укладку вершин с очень большой шириной, т.е. числом вершин на одном уровне. В методе Коффмана-Грехэма (см. [5]) задана максимальная ширина укладки w, и вершины распределяются так, чтобы на каждом уровне оказалось не более w вершин. Получившаяся укладка оказывается не очень широкой, однако число w должно быть задано заранее. Еще один метод укладки, предложенный в [7], минимизирует суммарную длину всех ребер графа (под длиной ребра понимается разность номеров уровней его конца и начала). Однако эти методы могут распределять вершины по уровням очень неравномерно. Например, если в графе есть путь $v_1, v_2, \ldots, v_{n-1}, v_n$ для некоторого большого n и путь v_1, u, v_n, то вершины $v_1, \ldots, v_n$ распределяются по n уровням, а вершина u может попасть на тот же уровень, что и v_{n-1}. В таком случае ребро (v_1, u) имеет длину $n-1$, а ребро (u, v_n) — длину 1. Если же поместить u на тот же уровень, что и $v_{[n/2]}$, то длины ребер (v_1, u) и (u, v_n) будут отличаться не более чем на единицу. В данной статье предлагается алгоритм распределения вершин по уровням, который пытается по возможности не создавать ребер с сильно отличающейся длиной.

В разделе 1 определяются используемые понятия и описывается алгоритм для распределения вершин графа по уровням. Доказывается корректность предложенного алгоритма, а также устанавливается, что его временная сложность составляет $O(|V||E|)$. В разделе 2 приводятся результаты вычислительных экспериментов. Приведен пример укладки графа, полученной с помощью предложенного алгоритма, а также время работы для графов разного размера.

1. Алгоритм распределения вершин по уровням

Сначала приведем используемые нами терминологию и обозначения. Ориентированный граф — это пара $G = (V, E)$, где V — конечное множество вершин, а $E \subseteq V \times V$ — множество ребер. Путь в графе — это последовательность вершин $v_1, v_2, \ldots, v_n$ такая, что $(v_i, v_{i+1}) \in E$ для всех $1 \leq i < n$. Длина пути — это число ребер в нем (то есть $n-1$). Если $v_1 = v_n$, то путь называется циклом. Граф называется ациклическим, если в нем нет циклов. Граф называется слабо связным, если для любых двух вершин $u, v \in V$ существует путь из u в v в графе $G' = (V, E')$, где $E' = E \cup \{\, (x, y) \mid (y, x) \in E \,\}$. Вершина s называется источником, если в нее не входят ребра. Вершина t называется стоком, если из нее не выходят ребра. В ациклическом графе всегда есть хотя бы один источник и хотя бы один сток. Глубина вершины v — это наибольшая длина пути из какого-нибудь источника в v (обозначается $d(v)$). Высота вершины v — это наибольшая длина пути

из v в какой-нибудь сток (обозначается $h(v)$). В ациклических графах любая вершина имеет конечную глубину и высоту. Глубины и высоты всех вершин можно найти за линейное время (см. [1]). Через $l(v)$ обозначим длину самого длинного пути, проходящего через вершину v. Легко видеть, что $l(v) = d(v) + h(v)$. Порядок вершин $v_1, \ldots, v_n$ называется топологическим, если для каждого ребра $(v_i, v_j) \in E$ выполняется неравенство $i < j$.

Задача поуровневой укладки графа состоит в следующем. Задан ациклический граф $G = (V, E)$. Требуется разбить множество его вершин V на несколько подмножеств $L_1, L_2, \ldots, L_r$, $L_i \cap L_j = \varnothing$ при $i \neq j$, так чтобы для любого ребра $(u, v) \in E$ выполнялось следующее условие: если $u \in L_i$, а $v \in L_j$, то $i < j$. Множества L_i называются уровнями. Если $i < j$, то будем говорить, что L_i находится выше L_j, то есть ребра направлены сверху вниз. В процессе нахождения уровней мы полагаем $L(v) = i$, если вершина v уже отнесена к уровню L_i, а иначе $L(v) = 0$. Если для вершины v уже вычислен номер уровня $L(v)$, то будем говорить, что вершина v уложена. Если все вершины пути $v_1, \ldots, v_n$ уложены, то будем говорить, что путь уложен. Вычисление номера уровня вершины назовем укладкой вершины, а вычисление номеров уровней всех вершин пути — укладкой пути.

Мы будем рассматривать только слабо связные ациклические графы. Это не уменьшает общности, поскольку каждую связную компоненту можно уложить по отдельности. В слабо связных графах $|E| \geq |V| - 1$. Кроме того, мы будем рассматривать только графы с одним источником и одним стоком. Если граф G содержит несколько источников $s_1, \ldots, s_n$, то мы добавим новую вершину s и ребра $(s, s_1), \ldots, (s, s_n)$. Аналогично, если G содержит несколько стоков $t_1, \ldots, t_m$, то мы добавим новую вершину t и ребра $(t_1, t), \ldots, (t_m, t)$. Любая поуровневая укладка нового графа задает и некоторую укладку исходного графа.

Приведем общее описание алгоритма укладки. Сначала алгоритм находит самые длинные пути в графе. Если они содержат по n вершин, то пути укладываются на уровни $L_1, \ldots, L_n$. Все остальные вершины будут уложены на эти же уровни. Алгоритм находит самый длинный путь, вершины которого еще не распределены по уровням. Пусть путь $v_0, v_1, \ldots, v_k, v_{k+1}$ таков, что вершины v_0 и v_{k+1} уже попали на уровни L_i и L_j соответственно, а остальные вершины $v_1, \ldots, v_k$ еще не уложены. Далее алгоритм укладывает вершины $v_1, \ldots, v_k$ между уровнями L_i и L_j с шагом приблизительно равным $(j-i)/(k+1)$. Одновременно с этим проверяется, что никакое вновь уложенное ребро не будет направлено снизу вверх. Чтобы добиться этого, алгоритм перемещает вершины на другие уровни, стараясь не очень сильно отступить от «идеальных» положений. Описанный процесс продолжается до тех пор, пока все вершины не будут уложены.

Перейдем к детальному описанию алгоритма. Укладку самых длинных путей выполняет процедура EmbedLongestPaths. Заметим, что любой самый длинный путь обязательно содержит и источник, и сток, поэтому эти две вершины сразу будут уложены алгоритмом.

Прежде чем описать укладку более коротких путей, дадим некоторые вспомогательные определения.

Определение 1. *Пусть $G = (V, E)$ — ациклический ориентированный граф, $V' \subseteq V$ — произвольное множество вершин. V'-путь в графе G — это последовательность вершин $v_0, v_1, \ldots, v_k, v_{k+1}$ такая, что $v_0, v_{k+1} \in V'$, $v_i \notin V'$ для всех $1 \leq i \leq k$.*

Алгоритм 1. Укладка самых длинных путей.

EmbedLongestPaths(G)

Вход: ациклический ориентированный граф $G = (V, E)$.

1: Вычислить $d(v)$ и $h(v)$ для каждой вершины $v \in V$.
2: Пусть K — длина самого длинного пути в графе.
3: $W = \{ v \in V \mid d(v) + h(v) = K \}$;
4: **for all** $v \in W$ **do**
5: $\quad L(v) = d(v) + 1$;
6: **end for**
7: **for all** $v \in V \setminus W$ **do**
8: $\quad L(v) = 0$;
9: **end for**

Говоря неформально, V'-путь начинается в одной из вершин множества V', а затем проходит через вершины графа G, не принадлежащие V', до тех пор, пока не посетит вершину из V'. После этого путь прерывается. В алгоритме укладки в качестве V' будет выступать множество уже уложенных вершин. Тогда V'-путь — это путь, у которого уложены только первая и последняя вершины.

Определение 2. *Пусть $G = (V, E)$ — ациклический ориентированный граф, $V' \subseteq V$ — произвольное множество вершин, $v \in V \setminus V'$ — некоторая вершина.*

1. *V'-глубина вершины v — это длина самого длинного пути $v_0, v_1, \ldots, v_k$ такого, что $v_0 \in V'$, $v_i \in V \setminus V'$ для $1 \le i \le k$, $v_k = v$. V'-глубина вершины v обозначается $d_{V'}(v)$.*

2. *V'-высота вершины v — это длина самого длинного пути $v_0, v_1, \ldots, v_k$ такого, что $v_i \in V \setminus V'$ для $0 \le i \le k-1$, $v_k \in V'$, $v_0 = v$. V'-высота вершины v обозначается $h_{V'}(v)$.*

3. *Через $l_{V'}(v)$ обозначается длина самого длинного V'-пути, проходящего через v.*

Если снова выбрать в качестве V' множество уложенных вершин, то тогда $d_{V'}(v)$ — это наибольшее расстояние от уже уложенной вершины до v, $h_{V'}(v)$ — наибольшее расстояние от v до уже уложенной вершины, а $l_{V'}(v)$ — длина самого длинного еще не уложенного пути. Величины $d_{V'}(v)$, $h_{V'}(v)$ и $l_{V'}(v)$ связаны простым соотношением.

Лемма 1. *Для любой вершины v и любого множества $V' \subseteq V$ выполняется равенство $l_{V'}(v) = d_{V'}(v) + h_{V'}(v)$.*

Доказательство. Если $v_0, v_1, \ldots, v_k$ — путь в вершину $v = v_k$ длины $d_{V'}(v)$, а $v_k, v_{k+1}, \ldots, v_r$ — путь из вершины v длины $h_{V'}(v)$, то $v_0, \ldots, v_k, \ldots, v_r$ — V'-путь длины $d_{V'}(v)+h_{V'}(v)$, проходящий через v. Поэтому $l_{V'}(v) \ge d_{V'}(v)+h_{V'}(v)$. Предположим, что это неравенство строгое. Тогда существует V'-путь $v_0, v_1, \ldots, v_k, \ldots, v_{r-1}, v_r$, проходящий через вершину $v = v_k$ и имеющий длину строго большую $d_{V'}(v) + h_{V'}(v)$. Но тогда либо длина пути $v_0, \ldots, v_k$ больше чем $d_{V'}(v)$, либо длина пути $v_k, \ldots, v_r$ больше чем $h_{V'}(v)$, что невозможно по определению 2. □

Алгоритм для вычисления V'-глубин и V'-высот получается простой модификацией стандартного алгоритма поиска самых длинных путей в ациклических графах (см. [1]). Как и классический алгоритм поиска самых длинных путей, процедура CalculateDepths перебирает вершины графа в топологическом порядке и запоминает найденные более длинные пути. Отличие состоит только в том, что при достижении вершины $u \in V'$ она не пытается продолжать пути через u, а сохраняет для u нулевое значение глубины. Для вычисления V'-высот вершин достаточно перевернуть ребра графа и применить к получившемуся графу алгоритм CalculateDepths.

Алгоритм 2. Вычисление V'-глубин вершин графа.

CalculateDepths(G, V').
Вход: ациклический ориентированный граф $G = (V, E)$, множество $V' \subseteq V$.
Выход: значения $d_{V'}(v)$ для всех $v \in V$.

```
1: Пусть v_1, ..., v_n — вершины графа G, расположенные в топологическом порядке.
2: for i = 1 to n do
3:     d(v_i) = 0;
4: end for
5: for i = 1 to n do
6:     for all (v_i, v_j) ∈ E, для которых v_j ∉ V' do
7:         if d(v_j) < d(v_i) + 1 then
8:             d(v_j) = d(v_i) + 1;
9:         end if
10:    end for
11: end for
12: return d;
```

Алгоритм 3. Вычисление V'-высот вершин графа.

CalculateHeights(G, V').
Вход: ациклический ориентированный граф $G = (V, E)$, множество $V' \subseteq V$.
Выход: значения $h_{V'}(v)$ для всех $v \in V$.

```
1: E' = { (u, v) | (v, u) ∈ E };
2: G' = (V, E')
3: h = CalculateDepths(G', V');
4: return h;
```

Процедура FindLongestPath служит для поиска длинных V'-путей.

Алгоритм 4. Поиск длинных путей.

FindLongestPath$(G, V', d_{V'}, h_{V'})$

Вход: ациклический ориентированный граф $G = (V, E)$, множество $V' \subseteq V$, функции $d_{V'}$ и $h_{V'}$.

Выход: некоторый V'-путь максимальной длины.

1: $m = \max\{\, d_{V'}(v) + h_{V'}(v) \mid v \in V \setminus V' \,\}$;
2: $W = \{\, v \in V \setminus V' \mid d_{V'}(v) + h_{V'}(v) = m \,\}$;
3: $F_1 = \{\, (u, v) \in E \mid u \in V', d_{V'}(v) + h_{V'}(v) = m \,\}$;
4: $F_2 = \{\, (u, v) \in E \cap (W \times W) \mid d_{V'}(v) = d_{V'}(u) + 1 \,\}$;
5: $F_3 = \{\, (u, v) \in E \mid d_{V'}(u) + h_{V'}(u) = m, v \in V' \,\}$;
6: Выбрать вершины v_0, v_1 такие, что $(v_0, v_1) \in F_1$.
7: $i = 1$;
8: **while** существует ребро $(v_i, u) \in F_2$ **do**
9: $\quad v_{i+1} = u; i = i + 1$;
10: **end while**
11: Выбрать вершину v_{i+1} такую, что $(v_i, v_{i+1}) \in F_3$.
12: **return** $(v_0, v_1, \ldots, v_i, v_{i+1})$;

Лемма 2. *Процедура* FindLongestPath *возвращает самый длинный V'-путь в графе G за время $O(|E|)$.*

Доказательство. Сначала процедура определяет длину m такого пути с помощью функций $d_{V'}$ и $h_{V'}$ и выделяет множество вершин W, через которые проходят пути длины m. Число m и множество W вычисляются верно по лемме 1. Затем процедура находит уложенную вершину v_0, из которой ведет ребро в вершину $v_1 \in W$. Такие вершины всегда существуют, так как единственный источник уже уложен. Далее процедура спускается по ребрам, входящим в самый длинный путь, пока такие ребра есть. Условие $d_{V'}(v_{i+1}) = d_{V'}(v_i) + 1$ гарантирует, что самый длинный путь, проходящий через v_i, может проходить через ребро (v_i, v_{i+1}). Путь завершается ребром, ведущим в уложенную вершину. Такая вершина всегда найдется, так как в графе есть только один сток, а он уже уложен.

Инициализация в строках 1–7 выполняется за время $O(|E|)$. Цикл в строках 8–10 тоже срабатывает за время $O(|E|)$, так как каждое ребро просматривается не более одного раза. □

Теперь опишем укладку более коротких путей. Пусть $V' = \{v \in V \mid L(v) \neq 0\}$ — множество вершин, уже распределенных по уровням. Пусть в графе $G = (V, E)$ имеется путь $v_1, v_2, \ldots, v_k$, вершины которого еще не распределены по уровням. Выберем самый длинный из таких путей. В таком случае все вершины, из которых ведут ребра в v_1, и все вершины, в которые ведут ребра из v_k, уже распределены по уровням, так как в противном случае путь можно было бы продлить. Пусть $i = \max\{L(u) \mid (u, v_1) \in E\}$, $j = \min\{L(u) \mid (v_k, u) \in E\}$. Заметим, что i и j всегда определены. Действительно, если, например, i не определено, то v_1 является источником, но единственный источник уже уложен.

Все вершины $v_1, \ldots, v_k$ должны располагаться между уровнями L_i и L_j. Мы стремимся распределить вершины $v_1, \ldots, v_k$ по уровням так, чтобы расстояния между уровнями, т.е. числа $L(v_1) - i$, $L(v_{m+1}) - L(v_m)$, $j - L(v_k)$, отличались

как можно меньше. Сначала мы вычислим «идеальное» расположение вершин $v_1, \ldots, v_k$, игнорируя возможность того, что при таком расположении некоторые ребра могут оказаться направленными снизу вверх. Далее мы уточним алгоритм так, чтобы учесть направление всех ребер.

Алгоритм 5. Укладка пути $v_1, \ldots, v_k$.

EmbedPath$(G, v_0, v_1, \ldots, v_k, v_{k+1})$

Вход: ациклический ориентированный граф $G = (V, E)$, путь $v_0, v_1, \ldots, v_k, v_{k+1}$, в котором $L(v_0) \neq 0$, $L(v_{k+1}) \neq 0$, $L(v_m) = 0$ для $1 \leq m \leq k$.

1: $i = L(v_0); j = L(v_{k+1})$;
2: **for** $m = 1$ **to** k **do**
3: **if** $j - i$ делится на $k + 1$ **then**
4: $p = \dfrac{j-i}{k+1}; n_1 = i + mp$;
5: **else**
6: $p = \left[\dfrac{j-i}{k+1}\right]; x = \left(\left[\dfrac{j-i}{k+1}\right] + 1\right)(k+1) + i - j$;
7: **if** $m \leq x$ **then**
8: $n_1 = i + mp$;
9: **else**
10: $n_1 = i + xp + (m - x)(p + 1)$;
11: **end if**
12: **end if**
13: $n_2 = \min\{L(u) \mid L(u) \neq 0, (v_m, u) \in E\}$;
14: $n_3 = \max\{L(u) \mid L(u) \neq 0, (u, v_m) \in E\}$;
15: $L(v_m) = \max\{\min\{n_1, n_2 - 1\}, n_3 + 1\}$;
16: **for** $r = m$ **to** 2 **step** -1 **do**
17: **if** $L(v_r) \leq L(v_{r-1})$ **then**
18: $L(v_{r-1}) = L(v_r) - 1$;
19: **end if**
20: **end for**
21: **end for**

Предположим сначала, что вершины $v_1, \ldots, v_k$ можно расположить с равным шагом p. Тогда вершины попадут на уровни с номерами $i + p, i + 2p, \ldots, i + kp$. Кроме того, на уровне L_i расположена вершина v_0, а на уровне $L_j = L_{i+(k+1)p}$ — вершина v_{k+1}. Из равенства $j = i + (k + 1)p$ получаем $p = (j - i)/(k + 1)$. Итак, если $j - i$ делится на $k + 1$, то вершины нужно укладывать с шагом $(j - i)/(k + 1)$.

Теперь предположим, что число $p = (j - i)/(k + 1)$ не является целым. Обозначим через q целую часть числа p, то есть $q = [(j - i)/(k + 1)]$. В этом случае мы уложим несколько вершин с шагом q (обозначим их число через x), а остальные — с шагом $q + 1$ (обозначим их число через y). Тогда первые вершины (от v_0 до v_x) попадают на уровни $L_i, L_{i+q}, L_{i+2q}, \ldots, L_{i+xq}$, а оставшиеся вершины (от v_{x+1} до v_{k+1}) попадают на уровни $L_{i+xq+(q+1)}, L_{i+xq+2(q+1)}, \ldots, L_{i+xq+y(q+1)}, L_{i+xq+(y+1)(q+1)} = L_j$. Мы полу-

чили систему уравнений

$$\begin{cases} x+y &= k, \\ i+xq+(y+1)(q+1) &= j. \end{cases}$$

Решая ее, находим $x = \left(\left[\frac{j-i}{k+1}\right]+1\right)(k+1)+i-j$. Подчеркнем, что эта формула получена в предположении, что $j-i$ не делится на $k+1$.

Процедура EmbedPath распределяет по уровням вершины одного пути согласно описанному методу. Дополнительно в строках 13–15 проверяется, что вершина не окажется ниже достижимой из нее вершины или выше вершины, достижимой из v_m. Далее мы докажем, что если пути перебираются по убыванию длины, то всегда существует уровень, расположенный между L_{n_3} и L_{n_2}. Если «идеальный» уровень оказывается слишком низким, то процедура просто помещает вершину на самый низкий уровень, лежащий выше всех ее потомков. Если после этого вершина v_m оказалась выше вершины v_{m-1}, то процедура поднимает вершину v_{m-1}. Далее в случае необходимости так же поднимаются вершины $v_{m-2}, \ldots, v_1$.

Наконец, приведем основной алгоритм. Мы считаем, что номера уровней $L(v)$ являются глобальными переменными, доступными во всех процедурах.

Алгоритм 6. Укладка графа.

EmbedGraph(G)

Вход: ациклический ориентированный граф $G = (V, E)$.

1: Отсортировать вершины графа G в топологическом порядке.
2: EmbedLongestPaths(G);
3: $V' = \{v \in V \mid L(v) \neq 0\}$;
4: **while** $V' \neq V$ **do**
5: $\quad d =$ CalculateDepths(G, V');
6: $\quad h =$ CalculateHeights(G, V');
7: $\quad \pi =$ FindLongestPath(G, V', d, h);
8: $\quad$ Пусть $\pi = (v_0, v_1, \ldots, v_k, v_{k+1})$.
9: $\quad$ EmbedPath($G, v_0, v_1, \ldots, v_k, v_{k+1}$);
10: $\quad V' = \{v \in V \mid L(v) \neq 0\}$;
11: **end while**

Теорема 1. *Алгоритм* EmbedGraph *распределяет вершины ациклического ориентированного графа $G = (V, E)$ по уровням за время $O(|V||E|)$.*

Доказательство. Сначала процедура EmbedLongestPaths укладывает самые длинные пути. Остается доказать, что каждый вызов процедуры EmbedPath уложит соответствующий путь. Пусть на очередной итерации рассматривается путь $v_0, v_1, \ldots, v_k, v_{k+1}$, где вершины v_0 и v_{k+1} уже уложены на уровни L_i и L_j, а остальные еще не уложены. Тогда между вершинами v_0 и v_{k+1} имеется не менее k уровней, то есть $j-i-1 \geq k$. Действительно, предположим сначала, что существует уложенный путь из v_0 в v_{k+1}. Если бы если бы число уровней между L_i и L_j было меньше k, то все уже уложенные пути из v_0 в v_{k+1} имели бы длину менее $k+1$. Но путь $v_0, v_1, \ldots, v_k, v_{k+1}$ имеет длину $k+1$, так что он должен был быть уложен еще

раньше. Если же между v_0 и v_{k+1} нет уложенного пути, то число уровней также должно быть не менее k, так как в противном случае, добавив вершины $v_1, \ldots, v_k$, снова можно было получить путь длиннее уже уложенных. Далее, все уложенные вершины, достижимые из v_m для $1 \le m \le k$, лежат на уровнях с номерами $L_{i+m+1}, L_{i+m+2}, \ldots$. Если бы из v_m была достижима вершина $u \in L_r$ для $r \le i+m$, то в графе был бы путь $v_0, v_1, \ldots, v_m, \ldots, u$ длины не менее $m+1$. Поскольку $v_0 \in L_i$, $u \in L_r$, $r < i+m$, то путь $v_0, v_1, \ldots, v_m, \ldots, u$ длиннее любого уже уложенного пути из v_0 в u. Но это невозможно, так как длинные пути рассматриваются в алгоритме раньше коротких. Наконец, покажем, что для каждой вершины пути $v_1, \ldots, v_k$ существует допустимый уровень. Пусть как и в процедуре EmbedPath $n_2 = \min\{L(u) \mid L(u) \ne 0, (v_m, u) \in E\}$, $n_3 = \max\{L(u) \mid L(u) \ne 0, (u, v_m) \in E\}$. Тогда $n_2 - n_3 > 1$. Если $n_2 - n_3 \le 1$, то существуют ребра $(u, v_m), (v_m, w) \in E$, для которых $L(u) - L(w) \le 1$. Но тогда, вставив эти два ребра, мы получили бы более длинный уложенный путь, что невозможно. Из трех доказанных утверждений следует, что вершины $v_1, \ldots, v_k$ всегда можно уложить на уровни между L_i и L_j: достаточно поместить вершину v_m на уровень L_{n_3+1}. Индукцией по m докажем, что вызов процедуры EmbedPath укладывает вершины $v_1, \ldots, v_m$. При $m = 1$ для v_1 вычисляется некоторая допустимая позиция (допустимость гарантируют строки 13–15). Пусть вершины $v_1, \ldots, v_{m-1}$ уже уложены. Сначала v_m помещается на уровень такой, что все достижимые из v_m вершины лежат на уровнях с большими номерами. Если v_m оказывается ниже v_{m-1}, укладка v_m на этом заканчивается. В противном случае начинает работать цикл в строках 16–20. После того, как в этом цикле рассмотрены вершины $v_m, \ldots, v_r$, выполняются неравенства $L(v_m) > L(v_{m-1}) > \cdots > L(v_r)$, а все вершины, достижимые из $v_m, v_{m-1}, \ldots, v_r$, лежат на уровнях с большими номерами. Действительно, на очередном шаге вершина v_r перемещается на уровень над вершиной v_{r+1}, и все достижимые из v_r вершины остаются на более низких уровнях.

Теперь оценим время работы. Топологическая сортировка выполняется за время $O(|V|+|E|)$ (см. [1]). Укладка самых длинных путей также требует $O(|V|+|E|)$ времени. Цикл в строках 4–11 выполняет не более $|V|$ итераций. Вычисление глубин и высот, а также поиск V'-пути требуют $O(|V| + |E|)$ времени. Строка 10 выполняется за время $O(|V|)$. Поэтому время работы цикла без учета вызовов EmbedPath составляет $O(|V||E|)$. Наконец, оценим время вызовов процедуры EmbedPath. Если укладывается путь $v_0, \ldots, v_{k+1}$, то укладка одной вершины требует $O(k + |V|)$ времени, так как строка 13 требует просмотра не более чем $|V|$ вершин, а цикл в строках 15–19 выполняет не более k итераций. Поэтому укладка всего пути требует $O(k^2 + k|V|)$ времени. Значит, если рассматриваются пути с $k_1, k_2, \ldots, k_n$ вершинами, то общее время всех вызовов составляет $O(\sum\limits_{i=1}^{n} k_i^2 + |V|^2)$, так как $\sum\limits_{i=1}^{n} k_i \le |V|$. При таком условии $\sum\limits_{i=1}^{n} k_i^2 \le |V|^2$, так что суммарное время всех вызовов EmbedPath составляет $O(|V|^2)$. Следовательно, цикл 4–11 алгоритма EmbedGraph срабатывает за время $O(|V||E|)$, и общее время составляет $O(|V||E|)$. □

2. Вычислительные эксперименты

Описанный алгоритм распределения вершин по уровням был реализован на

C++ (стандарт C++11). Эксперименты были проведены на компьютере со следующими характеристиками: Intel(R) Core(TM) i5-2400-CPU, 3.10 GHz, 4 ядра, ОЗУ 32ГБ, операционная система Windows 7 64bit. На Рис. 1 показаны две укладки одного графа. Левая укладка была получена с помощью алгоритма, помещающего каждую вершину на уровень на единицу больший ее глубины. Правая укладка была получена с помощью алгоритма, описанного в статье. Видно, что предложенный метод распределил вершины по уровням более равномерно, а длины ребер отличаются незначительно.

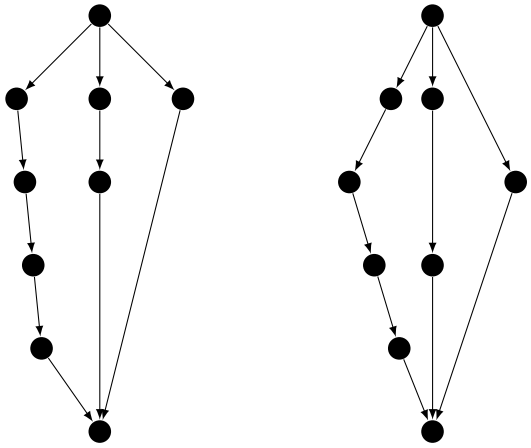

Рис. 1: Пример укладки графа

Из теоремы 1 следует, что время работы алгоритма быстро растет с ростом размера графа. Для плотных графов, т.е. для графов, в которых число ребер близко к максимально возможному, время работы составляет $O(|V|^3)$. Но если граф содержит очень много ребер, то при любом расположении вершин по уровням неизбежно значительное число пересечений. Это следует из известной теоремы о планарных графах: для любого планарного графа $G = (V, E)$ выполняется неравенство $|E| \leq 3|V| - 6$ (см. [2]). Поэтому интерес представляет время работы алгоритма для графов с небольшим числом ребер. В таблице 1 приведены результаты экспериментов для графов с плотностью 0,05 и 0,5. Под плотностью графа $G = (V, E)$ понимается отношение числа ребер графа к максимально возможному числу ребер, т.е. число $\frac{2|E|}{|V|(|V|-1)}$. Также приведено время работы алгоритма на графах, удовлетворяющих условию $|E| = 3|V|$. Для графов, содержащих 6000 вершин и миллион ребер, время работы составляет около 30 минут.

Заключение

В статье был предложен метод поуровневой укладки графов, распределяющий вершины по уровням так, чтобы длины ребер каждого пути отличались незначительно. Были проведены вычислительные эксперименты, которые показали, что

Таблица 1: Результаты экспериментов

Число вершин	$\|E\| = 3\|V\|$		Плотность $\approx 0,05$		Плотность $\approx 0,5$	
	Число ребер	Время (с)	Число ребер	Время (с)	Число ребер	Время (с)
100	300	<1	300	<1	3000	<1
1000	3000	2	25000	4	250000	28
2000	6000	15	100000	44	1000000	403
4000	12000	62	400000	444	4000000	6146
6000	18000	304	1000000	1830	15000000	>8 часов

для графов не очень большого размера с относительно маленькой плотностью время работы алгоритма является приемлемым для практического использования. Недостатком предложенного алгоритма является его высокая временная сложность $O(|V||E|)$. Хотя в практически важных случаях графы содержат относительно немного вершин и имеют не очень высокую плотность, представляет интерес разработка более эффективного варианта алгоритма. Также можно изучить возможность применения параллельных вычислений для одновременной укладки нескольких путей.

Список литературы

[1] Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ / под ред. И.В. Красикова. 2-е издание. М.: Издательский дом «Вильямс», 2005. 1296 с.

[2] Харари Р. Теория графов / под ред. Г.П. Гаврилова. М.: Едиториал УРСС, 2003. 296 с.

[3] Bastert O., Matuszewski C. Layered drawings of digraphs // In Kaufmann M., Wagner D. Drawing Graphs: Methods and Models. Lecture Notes in Computer Science, vol. 2025. Springer-Verlag, 2001. Pp. 87–120. https://doi.org/10.1007/3-540-44969-8_5

[4] Carpano M.J. Automatic Display of Hierarchized Graphs for Computer-Aided Decision Analysis // IEEE Transactions on Systems, Man, and Cybernetics. 1980. Vol. 10, № 11. Pp. 705–715. https://doi.org/10.1109/TSMC.1980.4308390

[5] Coffman E.G., Graham R.L. Optimal scheduling for two processor systems // Acta Informatica. 1972. Vol. 1. Pp. 200–213. https://doi.org/10.1007/BF00288685

[6] Eiglsperger M., Kaufmann M., Siebenhaller M. An efficient implementation of Sugiyama's algorithm for layered graph drawing // Journal of Graph Algorithms and Applications. 2005. Vol. 9, № 3. Pp. 305–325. https://doi.org/10.1007/978-3-540-31843-9_17

[7] Gansner E., Koutsofios E., North S., Vo K.-P. A technique for drawing directed graphs // IEEE Trans. Softw. Eng. 1993. Vol. 19, № 3. Pp. 214–230. https://doi.org/10.1109/32.221135

[8] Sander G. Graph layout for applications in compiler construction // Theoretical Computer Science. 1999. Vol. 217, № 2. Pp. 175–214. https://doi.org/10.1016/S0304-3975(98)00270-9

[9] Sugiyama K., Tagawa S., Toda M. Methods for visual understanding of hierarchical system structures // IEEE Transactions on Systems, Man, and Cybernetics. 1981. Vol. 11, № 2. Pp. 109–125. https://doi.org/10.1109/TSMC.1981.4308636

[10] Warfield J.N. Crossing theory and hierarchy mapping // IEEE Transactions on Systems, Man, and Cybernetics. 1977. Vol. 7, № 7. Pp. 505–523. https://doi.org/10.1109/TSMC.1977.4309760

Образец цитирования

Карлов Б.Н., Наймушин А.В. Равномерная поуровневая укладка графов // Вестник ТвГУ. Серия: Прикладная математика. 2018. № 2. С. 85–98. https://doi.org/10.26456/vtpmk496

Сведения об авторах

1. **Карлов Борис Николаевич**
доцент кафедры информатики Тверского государственного университета.

Россия, 170100, г. Тверь, ул. Желябова, д. 33, ТвГУ.
E-mail: bnkarlov@gmail.com.

2. **Наймушин Алексей Владимирович**
магистрант факультета прикладной математики и кибернетики Тверского государственного университета.

Россия, 170100, г. Тверь, ул. Желябова, д. 33, ТвГУ.
E-mail: alexnikn2@yandex.ru.

UNIFORM GRAPH LAYERING

Karlov Boris Nikolaevich
Associate professor at Computer Science department,
Tver State University
Russia, 170100, Tver, 33 Zhelyabova str., TverSU.
E-mail: bnkarlov@gmail.com

Naimushin Alexey Vladimirovich
Master student of Applied Mathematics and Cybernetics faculty,
Tver State University
Russia, 170100, Tver, 33 Zhelyabova str., TverSU.
E-mail: alexnikn2@yandex.ru

Received 28.05.2018, revised 22.06.2018.

In the article we study an algorithm for layering directed acyclic graphs. We propose a method for such layering of the vertices when the vertices of a path are placed on layers with approximately equal intervals. At first the described algorithm places on layers those vertices which are located on the longests paths of the graph. Then the algorithm places on the same layers remaining vertices going through the paths not yet embedded in the order from longer ones to shorter ones. In order to find long paths which are not yet embedded we use a modified method of searching for paths in acyclic graphs based on depth-first search and topological sorting. It is proved that time complexity of the described algorithm when working on a graph $G = (V, E)$ is $O(|V||E|)$. Computing experiments were performed which showed that for not very large graphs with relatively low density the running time of the algorithm is acceptable for practical use.

Keywords: graph embedding on plane, layer assignment of vertices, Sugiyama's algorithm, time complexity.

Citation

Karlov B.N., Naimushin A.V. Uniform graph layering. *Vestnik TvGU. Seriya: Prikladnaya Matematika* [Herald of Tver State University. Series: Applied Mathematics], 2018, no. 2, pp. 85–98. (in Russian). https://doi.org/10.26456/vtpmk496

References

[1] Cormen T., Leiserson C., Rivest R., Stein C. *Introduction to Algorithms.* 2nd ed. MIT Press and McGraw-Hill, 2001.

[2] Harary R. *Graph Theory.* Addison-Wesley, Reading, MA, 1969.

[3] Bastert O., Matuszewski C. Layered drawings of digraphs. In: *Drawing Graphs: Methods and Models*. Ed. by M. Kaufmann, D. Wagner. Lecture Notes in Computer Science, vol. 2025. Springer-Verlag, 2001. Pp. 87–120. https://doi.org/10.1007/3-540-44969-8_5

[4] Carpano M.J. Automatic display of hierarchized graphs for computer-aided decision analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, 1980, vol. 10(11), pp. 705–715. https://doi.org/10.1109/TSMC.1980.4308390

[5] Coffman E.G., Graham R.L. Optimal scheduling for two processor systems. *Acta Informatica*, 1972, vol. 1, pp. 200–213. https://doi.org/10.1007/BF00288685

[6] Eiglsperger M., Kaufmann M., Siebenhaller M. An efficient implementation of Sugiyama's algorithm for layered graph drawing. *Journal of Graph Algorithms and Applications*, 2005, vol. 9(3), pp. 305–325. https://doi.org/10.1007/978-3-540-31843-9_17

[7] Gansner E., Koutsofios E., North S., Vo K.-P. A technique for drawing directed graphs. *IEEE Trans. Softw. Eng.*, 1993, vol. 19(3), pp. 214–230. https://doi.org/10.1109/32.221135

[8] Sander G. Graph layout for applications in compiler construction. *Theoretical Computer Science*, 1999, vol. 217(2), pp. 175–214. https://doi.org/10.1016/ S0304-3975(98)00270-9

[9] Sugiyama K., Tagawa S., Toda M. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 1981, vol. 11(2), pp. 109–125. https://doi.org/10.1109/TSMC.1981.4308636

[10] Warfield J.N. Crossing theory and hierarchy mapping. *IEEE Transactions on Systems, Man, and Cybernetics*, 1977, vol. 7(7), pp. 505–523. https://doi.org/10.1109/TSMC.1977.4309760