**Д. Е. Бабурин**

# ИЕРАРХИЧЕСКИЙ ПОДХОД ДЛЯ АВТОМАТИЧЕСКОГО РАЗМЕЩЕНИЯ АЦИКЛИЧЕСКИХ ГРАФОВ*

## 1. ВВЕДЕНИЕ

В настоящее время графовое представление информации все больше и больше используется в различных областях точных и естественных наук. В программировании графовые модели применяются при создании программного обеспечения (управляющие графы, иерархии классов, диаграммы потоков данных), дизайне баз данных (диаграммы сущностей-связей), разработке информационных систем и систем реального времени (модели компьютерных сетей, графы состояний, сети Петри), а также в системном программировании — при создании теории компиляции и преобразования программ. Вопрос визуализации подобных графовых структур является краеугольным камнем в процессе адекватного отображения информации [2, 8, 55].

Автоматическое размещение графов — это задача конструирования геометрического представления графа, удовлетворяющего набору эстетических критериев. В последние пятнадцать лет эта проблема подвергается интенсивному изучению. За это время было опубликовано большое количество работ, начиная с 92-го года проводятся ежегодные конференции. Тем не менее задача автоматического размещения графов так и остается до конца не решенной, в том смысле, что до сих пор не было предложено универсального метода ее решения.

Возникшую ситуацию можно объяснить тем, что алгоритм размещения должен «учитывать» такие разносторонние аспекты, как: класс графа, семантика графовой информации, соглашения об изображении, эстетические критерии и ограничения на получаемое изображение. Перечисленные аспекты настолько многогранны, и их выбор настолько сильно зависит от случая конкретного применения, что разработка универсального алгоритма на данном этапе не представляется возможной.

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

В связи с данной проблемой был разработан ряд подходов к автоматическому размещению графов, которые базируются на фиксации некоторого набора из указанных критериев/аспектов. Так, например, был разработан целый ряд методов, позволяющих разместить планарный граф с прямолинейными или ортогональными дугами, или методы, основанные на физической упруго-силовой модели, которые больше подходят для сильно связанных неориентированных графов с прямолинейными ребрами.

Далее мы сконцентрируем свое внимание на ациклических графах. Выбор нами этого подкласса можно объяснить двумя причинами. Во-первых, преимущественное большинство реальных графов, встречающихся в программировании, являются ациклическими, а во-вторых, любой ориентированный (и тем более неориентированный) граф может быть преобразован к ациклическому путем смены или задания ориентации у части его ребер. Подобные преобразования будут нами подробно обсуждены в пятом разделе данного обзора.

При размещении ациклических графов во внимание принимаются следующие эстетические критерии:

1) совпадающее направление ребер;
2) минимизация площади изображения;
3) равномерность распределения вершин по площади всего изображения;
4) отсутствие слишком длинных ребер;
5) минимизация количества пересечения ребер;
6) прямолинейность ребер.

Одновременная оптимизация всех вышеперечисленных критериев является невозможной потому, что эти критерии зачастую оказываются противоречащими друг другу. Так, например, условие прямолинейности ребер нарушает равномерность распределения вершин и увеличивает общую площадь изображения.

Существует несколько различных техник размещения ациклических графов. Каждая из этих техник ставит во главу угла один или несколько из вышеперечисленных эстетических критериев и старается в меру возможностей удовлетворить оставшиеся критерии.

С одной стороны, можно решать задачу размещения ациклических графов сведением их к планарному графу и дальнейшему построению выпуклого изображения, используя особенности топологии планарных графов, с последующим восстановлением изначальной структуры. Так, например, хорошо изучена задача построения выпуклого изображения для планарных ациклических *st-графов*, т.е. графов, имеющих одну входную и одну вы-

ходную вершины, а также планарную укладку, в которой дуга, соединяющая эти вершины, лежит во внешней грани. Они допускают монотонное изображение с прямолинейными ребрами, внешняя грань которого есть заданный треугольник. Изображение в этом случае строится рекурсивно, путем выделения специальным образом st-подграфов с меньшим множеством вершин, чем у исходного, и применения алгоритма к ним (см. например [1, 18]). При таком подходе, когда фиксируется внешняя грань и дуги изображены прямыми линиями, с возрастанием степеней вершин углы между смежными ребрами быстро уменьшаются, что сильно понижает читаемость изображения.

В [28] описан метод сведения произвольного планарного ациклического графа к st-графу с сохранением планарности, однако само требование планарности исходного графа является слишком жестким. Задача планаризации произвольного ациклического графа является NP-трудной [46] (под планаризацией мы здесь понимаем задачу удаления минимального количества ребер так, чтобы сделать граф планарным). Даже несмотря на обилие существующих эвристик для решения этой задачи, планаризация не слишком подходит нашим целям, поскольку нарушает саму структуру исходного графа.

Задачу автоматического размещения ациклических графов можно также решать с помощью ортогональных техник, которые широко используются для графов произвольного вида и имеют хорошие временные показатели [12]. В этом случае при рассмотрении графов степени вершин более 4-х возникает проблема с появлением разброса в размерах вершин. Дело в том, что вершины с большой степенью размещают свои ребра за счет увеличения размеров, это приводит к одновременному наличию и слишком больших, и совсем маленьких вершин. Здесь главным недостатком, с нашей точки зрения, является то, что мы теряем возможность отражения иерархичности графа, к тому же размещения, полученные с использованием ортогонального подхода, более разряжены, чем, например, планарные размещения, и, соответственно, имеют большую площадь, что понижает удобность изучения структуры графа.

Подход, описанный в [72], позволяет визуализировать произвольный ациклический граф методом его декомпозиции на кланы — подграфы, состоящие из имеющих общие связи с остальной частью графа вершин. Такой метод хорошо отражает структуру графа, но имеет неудовлетворительное качество проведения ребер. Изначально прямолинейные ребра могут пересекать вершины графа, а эффективность введенных эвристик для устранения таких случаев путем создания дополнительных изломов ребер тео-

ретически не обоснована. Похожий подход, основанный на принципе «разделяй и властвуй», описан в [60]. Детальный обзор по вопросам кластеризации и размещения иерархических графовых структур может быть найден в [7].

Наиболее же известную идею размещения ациклических графов можно рассматривать как обобщение случая размещения деревьев. Для подчеркивания иерархичности структуры используются, как и в древесном случае, поуровневые представления, в которых все дуги следуют одному и тому же направлению [66]. Такие представления называются монотонными поуровневыми представлениями.

Настоящая работа посвящена обзору такой методологии (иерархический подход), применяемой для размещения ациклических графов. Данный подход является крайне интуитивным и позволяет для произвольного ациклического графа построить монотонное поуровневое представление с ребрами в виде ломанных или сплайнов. Первоначально идея была сформулирована еще в 1981 году [74], а также в некоторых других очень близких по тематике работах [77, 20]. Дальнейшее развитие (см. аннотированную библиографию [9]) сохранило основную концепцию, в которой иерархический подход содержит три части:

1) распределение вершин по уровням так, чтобы все дуги следовали одному направлению;
2) выбор порядка вершин на уровне с целью минимизации пересечений ребер;
3) определение координат вершин на уровне с целью минимизации общей длины ребер и количества изломов.

Следует отметить, что данный подход оперирует различными эстетическими критериями на каждом из своих шагов.

Для успешного применения иерархического подхода система изображения графов (исходные данные, визуализирующая компонента, область применения) должна удовлетворять следующим ограничениям:

1) исходный граф должен быть ациклическим мультиграфом или мультиграфом, «близким» к ациклическому. Близость здесь следует понимать в том смысле, что смена ориентации малого количества дуг должна превратить граф в ациклический;
2) семантика графовой информации должна предполагать возможность построения монотонного изображения, подчеркивающего направленность или наличие некоторого порядка на множестве вершин;

3) система визуализации не должна предполагать проведение только прямолинейных ребер между вершинами графа. Ребра должны быть изображены в виде ломанных или сплайнов.

В следующем разделе нашего обзора приведено общее описание шагов иерархического подхода. Далее каждый их этих шагов подробно описан в отдельном разделе. Разд. 5 посвящен вопросам предварительного преобразования структуры графа. Общая структура и стиль данного обзора соответствует [10] и [14].

## 2. ЭТАПЫ ИЕРАРХИЧЕСКОГО ПОДХОДА

Канонически метод состоит из трех последовательно применяющихся шагов, которые изображены на рис. 1.

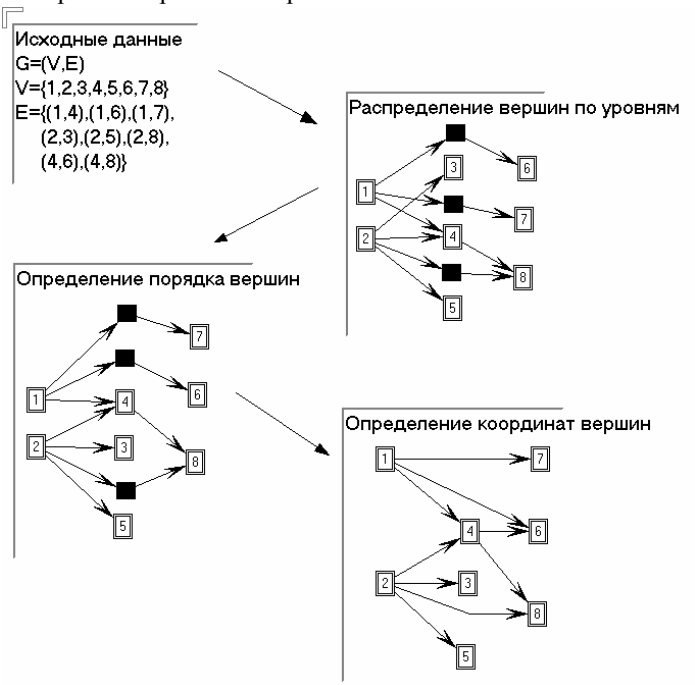


*Рис. 1.* Этапы иерархического подхода

- Распределение вершин по уровням. Каждой вершине u присваивается число L(u), указывающее уровень этой вершины так, чтобы все дуги, соединяющие вершины, следовали от меньшего номера к большему, при этом вершины одного уровня не должны быть связаны между собой. Подразумевается, что все вершины одного уровня будут расположены на одной прямой — вертикальной или горизонтальной — в зависимости от того, какое мы предпочитаем общее направление размещения: сверху вниз или слева направо. После проведения распределения вершин по уровням просматриваются все дуги: если дуга e = (u,v) проходит через несколько уровней, т.е. L(u) – L(v) > 1, то она удаляется из графа и заменяется на цепочку *фиктивных* вершин $v_1,\ldots,v_N$ такую, что $v_1 = u$, $v_N = u$, каждая вершина $v_i$ соединена дугой с $v_{i+1}$ и $L(v_{i+1}) = L(v_i) + 1$, а $v_1 = v$ и $v_N = u$.
- Определение порядка вершин на уровне. Задача данного шага состоит в сортировке вершин на каждом из уровней. Порядок вершин определяет топологию конечного изображения и должен быть выбран с целью минимизации пересечений ребер графа.
- Определение координат вершин на уровне. На данном шаге каждой вершине присваивается вертикальная координата (при размещении в горизонтальном направлении) так, чтобы сохранить порядок вершин на уровне, который был вычислен на предыдущем шаге. Координата выбирается с целью минимизации общей длины ребер и количества изломов. Наконец, конечное изображение получается представлением каждого ребра в виде прямолинейного отрезка и удаления фиктивных вершин. В этом случае длинные ребра оказываются изображенными в виде ломанных.

Некоторые исследователи склонны выделять задачу проведения ребер в отдельный этап алгоритма. В этих случаях ребра изображаются не каноническими ломанными, а, например, сплайнами [40, 67]. Для проведения сплайнов, не пересекающих вершины графа и проходящих через заданные области плоскости, существует алгоритм, описанный в [27].

Также задача проведения ребер выделяется в отдельный этап, если вершины графа имеют неточечное представление [68]. При этом приходится следить за отсутствием пересечений ребер и вершин графа, а также определять точки на изображении вершины, в которые будут входить и выходить ребра, инцидентные этой вершине графа.

Как уже было сказано выше, в случае, когда исходный граф не является ациклическим, требуется отдельный шаг алгоритма, который отвечает за

препроцессирование графа и преобразование его к ациклическому ориентированному графу. После построения изображения структура изначального графа восстанавливается.

Проблема оптимальной расстановки меток на дугах графа применительно к иерархическому размещению рассмотрена в [51].

Иерархический подход оказывается не очень удачным с точки зрения построения инкрементального размещения. Этот факт можно объяснить тем, что топология получаемого изображения очень сильно зависит от выделенной на первом шаге алгоритма иерархии вершин. А даже при локальных изменениях графа, как-то удаление или добавление вершин, групп вершин или ребер, не удается сохранить эту иерархию неизменной. Несмотря на это, в работах [16, 63, 22] рассмотрены разные подходы к инкрементальному размещению ациклических графов.

Вопросы инкрементального размещения очень тесно переплетены с вопросом удовлетворения ограничений на получаемое изображение. В качестве ограничений, которые могут быть удовлетворены при таком подходе, можно назвать требование разместить две выделенные вершины с одного уровня иерархии как можно ближе друг к другу, а также выравнивание вертикальных координат вершин с разных уровней иерархии [16, 45].

Иерархический подход размещает граф по уровням, образуя монотонное изображение. Направление следования уровней может быть как горизонтальным, так и вертикальным. В реальных системах это должно определяться семантикой изображаемой информации. Так, например, для иерархий классов более привычной является вертикальная укладка, для графов вызовов — горизонтальная. Иногда в данном вопросе решающее значение играет визуализирующая компонента или какие-либо ограничения системы. При горизонтальной укладке преимущественным направлением дуг является горизонтальное, что позволяет размещать на них длинные надписи. Поскольку данный обзор не затрагивает вышеприведенных вопросов, то будем считать, что уровни размещаются слева направо.

## 3. РАСПРЕДЕЛЕНИЕ ВЕРШИН ПО УРОВНЯМ

Задачей данного шага является присваивание каждой вершине ее конечной горизонтальной координаты. Для этого исходный ациклический граф $G = (V, E)$ должен быть приведен к *поуровневому* ациклическому графу. Поуровневое разбиение есть разделение V на подмножества $L_1, L_2, \ldots, L_h$, так что для каждого ребра $(u, v) \in E$, где $u \in L_i$ и $v \in L_j$, верно то, что i

> j. *Высотой* разбиения будем называть количество уровней *h*, а *шириной* w — число вершин на самом большом уровне. *Зазором* ребра (u, v), где $u \in L_i$ и $v \in L_j$, назовем разницу i – j. Разбиение называется *сжатым*, если не существует ребра с зазором, большим единицы.

После разбиения вершин по уровням всем вершинам одного уровня присваивается одна горизонтальная координата, т.е. $V_x = i$ для всех вершин с уровня $L_i$.

### 3.1. Техника фиктивных вершин

*Рис. 2*. Пример графа, не имеющего сжатого разбиения

Существование поуровневого распределения для ациклических графов очевидно. Однако не для всякого ациклического графа существует его сжатое разбиение — это может быть продемонстрировано уже для графа, состоящего всего из трех вершин (рис. 2). Необходимость же построения сжатого разбиения диктуется следующими двумя причинами. Во-первых, сжатое разбиение минимизирует горизонтальную длину ребер и количество их изломов. При сжатом разбиении ребра связывают вершины только соседних уровней и могут быть изображены прямолинейными отрезками. Во-вторых, сжатое разбиение существенно упрощает задачу минимизации пересечений ребер графа, которая решается на втором шаге алгоритма. Сжатое разбиение сводит эту глобальную задачу к задаче минимизации пересечений для двух соседних уровней.

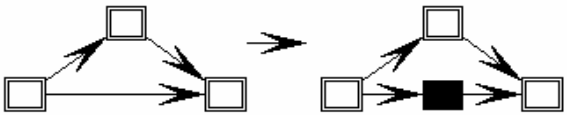

*Рис. 3.* Вставка фиктивных вершин

Для построения сжатого разбиения произвольного ациклического графа применяется техника *фиктивных* вершин. Фиктивные вершины добавляются в граф вдоль длинных ребер, т.е. ребер с зазором, большим единицы (рис. 3). Для каждого такого ребра (u, v), $u \in L_i$ и $v \in L_j$, с зазором $k = i - j > 1$ добавляется k – 1 вершин $u_1, \ldots u_{k-1}$ таких, что $u_m \in L_{i-m}$.

Такое построение приводит к тому, что в графе остаются только ребра, соединяющие вершины соседних уровней, и задача минимизации пересечения ребер локализуется до двухуровневого случая. Данное построение за-

одно решает и задачу проведения ребер в конечном изображении. Ребро изображается в виде ломанной, концами которой являются сами вершины, а точками излома — вставленные на первом этапе фиктивные вершины. Таким образом исчезает возможность пересечения ребер с вершинами.

Заметим здесь, что существуют работы по размещению ациклических графов, находящиеся в рамках данной методологии, однако не использующие технику фиктивных вершин [4]; в этих случаях дополнительное внимание уделяется вопросу проведения ребер графа.

### 3.2. Критерии построения разбиения

Постараемся сформулировать набор критериев, которые стоит принимать во внимание при построении поуровневого разбиения.

Искомое размещение графа должно быть компактно. Поскольку площадь фактически зависит только от количества уровней и максимального количества вершин на одном уровне, то компактизации можно добиться уменьшением одной из этих величин. При этом высота графа оказывается ограниченной снизу длиной максимального пути в графе. Существуют простые методы, позволяющие достичь нижней оценки высоты разбиения (п. 3.3), однако минимизация ширины есть NP-трудная задача, которая в реальных системах может быть решена только приближенно (п. 3.4). В связи с этим различием необходимо заметить, что предпочтение выбора величины для минимизации зачастую определяется на уровне визуализирующей системы. Если вершины графа изображаются не просто точками, а областями на плоскости, причем имеющими некоторые текстовые надписи, то при горизонтальной укладке наиболее важной становится высота разбиения, а при вертикальной — ширина. Это может быть еще одним хорошим доводом для использования горизонтального следования уровней в изображении графа.

Следующий важный критерий к разбиению вершин — это минимизация количества фиктивных вершин. Такое желание связано с тем, что введение фиктивных вершин повышает общую сложность алгоритма. Если мы рассматриваем достаточно плотный граф, в котором длинных ребер $O(N)$, то после их удаления мы получим $O(N^2)$ фиктивных вершин. Верхняя оценка числа вставленных фиктивных вершин может быть найдена в [36]. Немаловажно также и то, что чем больше мы вводим фиктивных вершин, тем больше получится в конечном размещении изломов на дугах. Таким образом, кроме минимизации общего числа вершин необходимо минимизировать максимальную длину ребра, то есть число фиктивных вершин, заме-

няющих одну дугу, потому как чем длиннее ребро и чем больше на нем изломов, тем сложнее при изучении изображения проследить соответствующую связь в графе. Ну и поскольку сложность всех шагов алгоритма размещения графа зависит от общего количества вершин, то мы должны стремиться к тому, чтобы не слишком увеличить это количество при добавлении фиктивных вершин. Способ построения разбиения с минимизацией количества фиктивных вершин рассмотрен в п. 3.5. К сожалению, одновременная минимизация высоты разбиения и количества фиктивных вершин уже оказывается NP-трудной задачей [53].

Поуровневое размещение иногда должно учитывать ограничения, налагаемые семантикой исходного графа. Так, в некоторых приложениях часть вершин оказывается предраспределенной по уровням. Это предраспределение может быть вызвано желанием выделить некоторые вершины или явно указать последовательность появления вершин (time-lines). Чаще всего такая задача может быть сведена к исходной путем предварительного слияния вершин с одинаковым предопределенным рангом.

### 3.2. Распределение по длиннейшему пути

Данный способ позволяет за линейное время распределить вершины графа по уровням так, чтобы высота получающегося разбиения оказалась минимальной. Для этого надо поместить все *стоки* (вершины без выходящих дуг) на уровень $L_1$, а каждую оставшуюся вершину — на уровень $L_{p+1}$, где p — длина наибольшего пути от этой вершины до ближайшего стока.

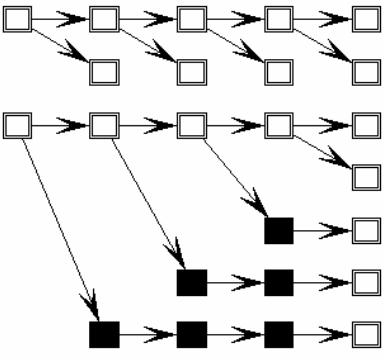

*Рис. 4*. Способы распределения вершин по уровням

Здесь следует отметить, что данный способ построения разбиения является инвариантным относительно направления дуг. При построении разбиения от стоков мы получим граф со всеми стоками, расположенными на одном уровне, но аналогичным образом разбиение можно построить и от *источников* (вершин без входящих дуг), и при этом все источники будут расположены на одном уровне. Очевидно, что высота обоих разбиений будет одинакова, однако ширина может отличаться на сколь угодно большую величину (рис.

4). Поскольку данный алгоритм является линейным, то представляется разумным построить оба разбиения и выбрать из них наименее широкое.

Основной недостаток распределения по длиннейшему пути — это отсутствие оптимизации на ширину разбиения. Проблемам минимизации ширины разбиения посвящен следующий раздел.

### 3.4. Минимизация ширины разбиения

К сожалению, задача нахождения разбиения с минимальной шириной и при этом ограниченной высотой является NP-полной [38]. NP-полнота задачи следует из сведения этой задачи к известной NP-полной задаче составления расписания для мультипроцессора. Каждая вершина ациклического графа представляет работу, которая может быть выполнена на любом из процессоров за единичное время. Ребро графа представляет ограничение предшествования на выполняющиеся работы. Составление расписания для мультипроцессора есть распределение работ по $W$ процессорам так, чтобы все они могли быть выполнены за время, не превосходящее $H$. Понятно, что такое расписание существует только в том случае, если есть разбиение графа по уровням с высотой, меньшей $H$, и шириной, меньшей $W$.

Эквивалентность задачи поуровневого разбиения задаче составления расписания для мультипроцессора позволяет сразу же воспользоваться результатами, достигнутыми в этой области. Существует ряд эвристик для решения задачи составления расписания, и эти эвристики могут быть применены к нашей задаче. Например, эвристика Кофман-Графама [24] позволяет построить разбиение ширины $W$, и при этом его высота не будет превосходить $(2-2/W)h_{min}$, где $h_{min}$ — минимально возможная высота разбиения шириной $W$. Данная оценка на высоту разбиения получена в [54].

Стоит заметить, что минимизируемая данными способами ширина разбиения может оказаться все равно не удовлетворительной, поскольку алгоритм не учитывает последующую вставку фиктивных вершин. В полученном графе могут оказаться уровни с огромным количеством ребер, идущих через них, и эти ребра будут заменены фиктивными вершинами, что расширит такие уровни. Однако, учитывая то, что в конечном изображении графа фиктивные вершины будут обратно заменены ребрами, можно сказать, что данный способ построения разбиения является вполне приемлемым, если ширина дуг графа мала по сравнению с размерами вершин. В противном случае предлагаются модификации эвристики Кофман-Графама, учитывающие ширину, «приносимую» фиктивным вершинами.

### 3.5. Минимизация количества фиктивных вершин

Существует полиномиальный алгоритм, позволяющий построить разбиение с минимальным количеством фиктивных вершин [40]. Нахождение оптимального разбиения формулируется в виде задачи линейного целочисленного программирования: $Min \sum_{(u,v)\in E} L_U - L_V$, где (u, v) — ребро с ограничениями: $L_U - L_V \geq 1$ и $L_U, L_V \geq 1$.

Существует ряд способов решения данной задачи за полиномиальное время. Первый — решить эквивалентную линейную задачу, а затем за полиномиальное время трансформировать решение в целочисленное. Другой способ — это сведение данной задачи к задаче минимизации потоков, для которой существуют полиномиальные решения [42].

Поскольку в такой постановке матрица ограничений является унимодулярной, то для решения может быть применен симплекс-метод, который хотя и не полиномиален, но на практике является быстросходящимся [40].

## 4. ОПРЕДЕЛЕНИЕ ПОРЯДКА ВЕРШИН НА УРОВНЕ

После построения поуровневого разбиения встает задача определения порядка расположения вершин на каждом уровне нашего графа. Нахождение такого распределения с целью минимизации количества пересечения ребер и есть задача данного этапа алгоритма.

В первую очередь стоит отметить, что количество пересечения ребер в поуровневом графе не зависит от конечных координат вершин, а зависит только от их относительного положения внутри каждого уровня. Таким образом, задача данного этапа является не геометрической, а всего лишь комбинаторной. Однако эта задача является NP-полной уже для графа, имеющего всего лишь два уровня [39], и, более того, NP-полной эта задача остается, даже если есть всего лишь одна вершина со степенью, большей единицы на каждом из уровней [59].

Задачу минимизации пересечений ребер, как в случае двух уровней [48, 76], так и глобальную [47, 44], естественно сформулировать в виде задачи целочисленного программирования. Однако, несмотря на возможность получения точного решения, такой подход может быть применен только для очень маленьких графов. Практические реализации показывают, что в разумное время решение задачи может быть найдено лишь для графов, содержащих не более 15 вершин (в случае глобальной оптимизации) и не бо-

лее 60 вершин для каждого из уровней (в случае минимизации пересечений лишь для двух соседних уровней).

В абсолютном большинстве известных автору работ, проблема минимизации пересечений ребер в поуровневом графе решается не глобально для всего графа, а лишь локально — для всех пар соседних уровней. Эксперименты [49] показывают, что сведение задачи минимизации пересечений ребер к случаю двух уровней дает результаты, очень далекие от оптимальных. Более приемлимых результатов минимизации глобального количества пересечений добиваются путем многократного «просматривания» всех соседних уровней и минимизации пересечений на них. Такая эвристика имеет несколько модификаций, которые различаются по следующим критериям:

- выбор способа фиксации уровней. При рассмотрении двух смежных уровней положение вершин на одном из уровней можно считать фиксированным и выбирать оптимальное расположение вершин другого уровня. Возможно также фиксирование положения вершин уровней $L_{i-1}$, $L_{i+1}$ при рассмотрении уровня $L_i$;
- выбор порядка, в котором просматриваются пары соседних уровней. Уровни могут просматриваться слева направо или наоборот, возможно также чередование направлений просмотра;
- выбор критериев окончания просмотров. Это может быть фиксированное количество просмотров или же просмотры делаются, пока они уменьшают или существенно уменьшают общее количество пересечений.

Отдельно стоит поднять вопрос о выборе первоначального порядка вершин на каждом из уровней. Традиционно для этого применяется метод [40], позволяющий избежать пересечения ребер в случае, если исходный граф был деревом. Метод состоит в проведении поиска в глубину, при котором вершины получают номера на своем уровне в порядке их просмотра в ходе такого поиска.

Итак, в дальнейшем мы будем рассматривать только задачу минимизации количества пересечений для двух уровней. Пусть мы имеем двудольный граф $G = (L_1, L_2, E)$, где $L_1$ — множество вершин первого уровня, $L_2$ — вершины второго уровня, E — множество ребер. На основе фиксированного расположения вершин первого уровня $x_1$ строится расположение вершин другого уровня. Как уже было сказано, данная задача является NP-полной, и для нахождения ее приближенного решения используются эвристики, описанные ниже.

### 4.1. Методы, основанные на сортировке

Данная группа эвристик основана на идее сортировки вершин уровня $L_2$ так, чтобы уменьшить количество пересечений ребер. Легко видеть, что для двух выбранных вершин $u \in L_2$ и $v \in L_2$ количество взаимопересечений инцидентных им ребер зависит только от относительного расположения этих двух вершин на уровне $L_2$ и не зависит от расположения всех остальных вершин этого уровня. Таким образом, между вершинами одного уровня можно установить отношение/порядок, определяемое количеством пересечений ребер, порождаемых в различных конфигурациях расположения вершин.

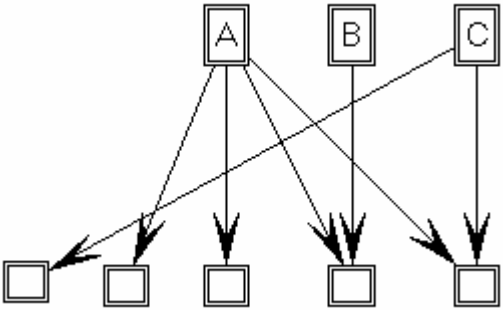


*Рис. 5*. Определение набора отношений между вершинами A, B и C

Понятно, что заданное таким образом отношение не может образовывать отношение линейного или частичного порядка. В противном случае обычная сортировка вершин решила бы нашу NP-полную задачу по крайней мере за квадратичное время. Иллюстрирует данное замечание следующий пример. На рис. 5 изображены два уровня иерархии с целью определения набора отношений между вершинами A, B и C. Легко видеть, что $A < B$, поскольку при расположении вершины A левее вершины B получается меньше пересечений ребер (одно против двух), $B = C$ (одно пересечение в любом из случаев), но, в свою очередь, $C < A$ (три пересечения против четырех). Получившаяся цепочка $A < B = C < A$ показывает, что заданное таким образом отношение не является транзитивным.

Однако закон трихотомии (для пары (a, b) справедливо одно и только одно из соотношений $a < b$, $a = b$, $a > b$) для нашего отношения выполняется. А это означает, что возможна устойчивая сортировка вершин, т.е. сортировка, при которой выполняются условия упорядоченности и устойчивости.

Выполнение такого упорядочивания и составляет суть методов, основанных на сортировке. Такие методы должны быстро считать потенциальное количество пересечений. Это может быть сделано напрямую за $O(|E|^2)$, а незначительными усилиями убыстрено до $O(\sum C_{uv})$, где $C_{uv}$ — количество пересечений, образуемое парой вершин $u$ и $v$. Подобный подсчет возможных пересечений для каждой пары нужно выполнить лишь один раз — это и задаст порядок на множестве вершин. Дальнейшая сортировка не изменит установившихся между вершинами отношений.

Первый из таких алгоритмов похож по своей идее на метод пузырька. Предлагается просматривать поочередно все пары соседних вершин и менять их местами, если при этом уменьшается количество пересечений. Процесс заканчивается, когда после очередного полного просмотра всех вершин второго уровня количество пересечений не уменьшилось (неупорядоченная пара не была найдена). Сложность данного алгоритма — $O(|L_2|^2)$, так как для конечного упорядочивания может потребоваться вплоть до $|L_2|$ просмотров. Результат работы такого алгоритма сильно зависит от выбора начального расположения, так как оперирует перестановками только в терминах соседних вершин.

Другой метод, позволяющий упорядочить множество вершин, основан на идее быстрой сортировки. Сначала выбирается «средняя» вершина, а все остальные вершины разбиваются на два множества в зависимости от того, в каком отношении они находятся с выбранной вершиной. Затем подобная процедура рекурсивно применяется к каждому из получившихся множеств. В худшем случае временная сложность данного алгоритма так же, как и алгоритма быстрой сортировки, является квадратичной, однако в среднем алгоритм требует всего лишь $O(|L_2| \log|L_2|)$ операций.

Идея следующего метода тоже использует концепцию сортировки вершин на уровне, хотя явно не прибегает при этом к заранее установленному отношению порядка на множестве вершин. Метод под названием «отсев» (Sifting) [62] на каждом шаге берет одну из вершин и пытается ее вставить в такое место, чтобы породить при этом наименьшее количество пересечений ребер. Для этого вершину пытаются последовательно вставить в каждое из возможных мест и считают количество получившихся пересечений. Поскольку при сдвиге вершины на одну позицию изменение количества пересечений можно посчитать за константное время (используя предварительно накопленную информацию), то алгоритм имеет квадратичную временную сложность и по своей сути похож на сортировку вставками.

Как уже было замечено, любой основанный на сортировке алгоритм упорядочивания вершин должен работать с набором отношений между па-

рами вершин, и вычисление этих отношений не может быть выполнено за линейное время. Поэтому любой из упомянутых методов будет тоже иметь нелинейную сложность.

Поскольку идеи упорядочивания заимствованы из общеизвестных методов сортировки, то авторство подобных методов чаще всего не указывается. Еще несколько методов, похожих на описанные, можно найти в [31].

### 4.2. Метод барицентров и метод медиан

На практике наиболее широко используются линейные алгоритмы, основанные на методе барицентров [74]. Координата вершины $v \in L_2$ определяется как среднее арифметическое координат всех ее связей с уровня $L_1$:

$$avg(u) = \frac{1}{\deg(u)} \sum_{v \in N(u)} x_1(v), \text{ где } N(u) := \{v \mid (u,v) \in E\}.$$

Если после перестановки координаты каких-либо двух вершин совпадают, то они разносятся на минимальное расстояние, порядок при этом определяется произвольно. Самая распространенная модификация этого метода — метод медиан [34], где вместо среднего арифметического используется координата среднего соседа с уровня $x_1$, т.е. если $u_1, u_2, \ldots, u_m \in L_1$ — вершины смежные v, причем $x_1(u_1) < x_1(u_2) < \ldots < x_1(u_m)$, то координата $x_2(v) = med(v)$ определяется как $x_1(u_{avg})$, где $avg = \lfloor m/2 \rfloor$. В случае, когда $med(v) = med(u)$ и v имеет нечетную степень, а u — четную, то v ставится перед u; если же четность степеней совпадает, то порядок выбирается произвольно. В работе [25] показано, что для каждой вершины u значение ее медианы может быть найдено за время $O(|N(u)|)$, что делает эту эвристику настолько же быстрой, как и метод барицентров.

Пусть $opt(G, x_1)$ — минимально возможное число пересечений ребер для данного графа, где $x_1$ — расположение вершин $L_1$, $avg(G, x_1)$ — число пересечений, остающихся после применения метода барицентров, $med(G, x_1)$ — число пересечений, остающихся после применения метода медиан. Приведем некоторые утверждения о методе барицентров и медиан (см. например [10]).

1. Если для данного графа G возможно размещение без пересечения ребер, то и метод медиан, и метод барицентров его находят, т.е. если размещение $x_1$ такое, что $opt(G, x_1) = 0$, тогда $avg(G, x_1) = med(G, x_1) = 0$.

2. Для любого n существует двудольный граф $G = (L_1, L_2, E)$, где $|L_1| = n$, $|L_2| = 2$, и размещение $x_1$ вершин $L_1$ такие, что $avg(G, x_1)/opt(G, x_1)$ пропорционально $n^{1/2}$.
3. Для любого n существует двудольный граф $G' = (L'_1, L'_2, E')$, где $|L'_1| = n$, $|L'_2| = 2$, и размещение $x'_1$ вершин $L_1$ такие, что $med(G', x'_1)/opt(G', x'_1) \geq 3$ — $O(1/n)$.
4. Для двудольного графа $G = (L_1, L_2, E)$ и любого размещения $x_1$ вершин $L_1$ верно неравенство $med(G', x'_1) \leq 2*opt(G', x'_1)$, если все вершины $L_2$ не имеют более трех ребер.

Известны также следующие модификации метода медиан.

1. *Средние медианы* [57]. Для вершин с четной степенью присваивается среднее арифметическое значение позиций вершин с соседнего уровня, и med(u) — для нечетных степеней.
2. *Полумедианы* [57]. Для вершин с четной степенью в качестве значения медианы берется значение, полученное по методу барицентров.
3. *Взвешенные медианы* [40]. Развитие метода средних медиан, отличающееся от него только в случае вершин с четной степенью, большей двух. В этом случае взвешенная медиана определяется как

$$vmed(u) = \frac{x_1(v_{j/2}) * right + x_1(v_{j/2+1}) * left}{left + right},$$

где $left = x_1(v_{j/2}) - x_1(v_1)$, и $right = x_1(v_j) - x_1(v_{j/2+1})$. Такая стратегия сдвигает вершины к той стороне, где ее соседи располагаются наиболее плотно.

### 4.3. Планаризация

Альтернативный подход представлен в [58]. Здесь автор предлагает заменить задачу отыскания перестановки вершин с целью минимизации пересечений ребер задачей планаризации двухуровневого ациклического графа. Под планаризацией понимается удаление минимального количества ребер, так чтобы полученный после этого двухуровневый граф можно было изобразить планарно. После нахождения необходимого минимального набора ребер граф изображается планарно, а удаленные ребра вставляются в получившуюся планарную укладку.

Понятно, что найденная таким способом перестановка вершин формально не решает задачу минимизации количества пересечений ребер, од-

нако интуитивно ясно, что полученное количество пересечений должно быть близко к минимально возможному. Причем такой алгоритм будет гарантировать достижение абсолютного оптимума в том случае, когда исходный двухуровневый граф допускает планарную двухуровневую укладку.

Аргументация такого подхода состоит в том, что визуальное восприятие запутанности графа и количества пересечений ребер зависит не только от самого числа пересечений, а также и от их взаимного расположения. Авторы подхода утверждают, что полученные таким образом представления выглядят зачастую даже лучше, чем представления, имеющие действительно минимальное количество пересечений ребер.

Несмотря на NP-полноту [35] такой постановки задачи планаризации двухуровневого графа, несомненным преимуществом перед задачей минимизации пересечений является наличие хороших эвристик ее решения с гарантированной оценкой худшего случая [75, 32].

В самой работе [58] задача планаризации двухуровневого ациклического графа формулируется в виде задачи линейного целочисленного программирования, позволяя для небольших графов находить ее точное решение.

### 4.4. Другие эвристики

*Вероятностная эвристика* [26]. Суть алгоритма заключается в «жадной вставке» вершин на основе предварительно подсчитанной для каждой вершины вероятности образования пересечений. Такая вероятность определяется суммой пересечений для каждого из ребер, инцидентных данной вершине, в полном двудольном графе.

*Алгоритм Тутти* [33]. Алгоритм состоит в предварительном определении позиций вершин на двух крайних уровнях с последующим решением системы разряженных линейных уравнений

$$x(u) = \frac{1}{2out\deg(u)} \sum_{v \in N^+(u)} x(v) + \frac{1}{2in\deg(u)} \sum_{w \in N^-(u)} x(w)$$

для определения координат вершин на всех промежуточных уровнях.

*Назначения* [21]. Данная эвристика позволяет свести задачу к известной задаче о назначениях. Для каждой вершины определяется число *dij* как верхняя оценка количества пересечений, образованных ребрами, инцидентными вершине *i*, в том случае, если вершину поместить на место j. Как и в

вероятностной эвристике *di* вычисляются путем дополнения графа до полного. Далее для матрицы $D = (dij)$ решается задача о назначениях, т.е. выбираются $n$ элементов так, чтобы покрыть ими каждую колонку и каждый столбец матрицы и при этом минимизировать сумму их значений.

*Генетический алгоритм* [61]. Авторами был сконструирован генетический алгоритм для двухуровневой задачи минимизации пересечений. Проведенное сравнение с барицентрическим методом показало, что такой алгоритм дает, как правило, лучшие решения, но является крайне медленным.

### 4.5. Замечания по поводу практического выбора метода

Сравнению различных методов минимизации пересечений посвящено сразу несколько работ [31, 56, 49, 62]. В основном такие работы проводятся каждым из исследователей при представлении нового метода минимизации пересечений. Здесь мы постараемся просуммировать имеющиеся результаты и дать рекомендации по практическому выбору метода решения этой задачи.

Во-первых, для графов с малой насыщенностью и малым количеством вершин возможен поиск оптимального решения за приемлемое время. В этих целях используются методы целочисленного программирования. Симплекс-метод, используемый для решения этой задачи, хоть и не имеет гарантированной полиномиальной сходимости, однако на практике оказывается весьма эффективным [49].

С другой стороны, результат [10] показывает, что для насыщенных графов минимальное число пересечений ребер очень близко к максимальному. Поэтому, начиная с некоторой степени насыщенности, представляется разумным отбросить все методы поиска наилучшего расположения вершин и вместо этого взять случайное расположение.

Таким образом, простор для применения эвристик возникает лишь в «средних» случаях. Как видно из приведенных выше утверждений, метод медиан теоретически выглядит привлекательнее метода барицентров в силу наличия гарантированной верхней оценки числа пересечений. Но на практике оба метода применяются с равным успехом, при этом после применения одного из барицентрических методов разумно применение методов, основанных на сортировке для локальной корректировки и окончательной доводки расположения вершин на уровне.

Отдельно стоит задуматься над критериями остановки итерационного эвристического процесса. Чаще всего остановка производится после проведения некоторого, заранее определенного, числа итераций, либо после

того, как конфигурация вершин перестанет изменяться, либо количество пересечений перестанет уменьшаться. Для методов, основанных на сортировке, выбрать количество необходимых итераций достаточно просто, однако эта величина не может быть разумно предопределена для барицентрических методов. Изменение конфигурации вершин на каждом из шагов алгоритма очень легко отследить в процессе вычислений, но практика показывает, что останов по такому критерию наступает крайне редко. Представленные методы склонны не иметь устойчивых конфигураций вершин.

Подсчет количества пересечений на каждом шаге алгоритма кажется самым объективным критерием останова. Проблема заключается в том, чтобы быстро вычислить это количество. Подсчет в лоб может быть произведен лишь за время $O(|E|^2)$. Для методов, основанных на сортировке, такой результат вполне приемлем, поскольку одновременно с подсчетом пересечений может быть выполнено и вычисление отношения порядка на множестве вершин. В работах [15, 23] представлен алгоритм, позволяющий вычислить порядок на множестве вершин и количество фактических пересечений за время $O(E*log(E + K))$, где $K$ — количество фактических пересечений. По сути же такой алгоритм остается квадратичным относительно числа ребер, поскольку в худшем случае количество пересечений квадратично.

Представленные квадратичные алгоритмы подсчета количества пересечений не могут быть приемлемыми для линейных барицентрических методов. Однако алгоритмы могут быть существенно убыстрены, если отказаться от одновременного вычисления отношения порядка на множестве вершин. Лучший из известных автору алгоритмов [78] производит вычисление количества пересечений ребер в двухуровневом графе за время $O(E*log(E))$, и, по всей видимости, этот результат не может быть существенно улучшен.

## 5. ОПРЕДЕЛЕНИЕ КООРДИНАТ ВЕРШИН НА УРОВНЕ

После определения порядка вершин на уровне необходимо определить их настоящие вертикальные координаты. Канонически, в иерархическом подходе, ребра графа изображаются в виде ломанных с точками излома, находящимися в фиктивных вершинах, поэтому задача определения окончательных координат всех вершин одновременно является и задачей проведения ребер. Если же ребра графа имеют какую-либо другую форму и/или

способ проведения, то соответствующие критерии должны быть рассмотрены при решении задачи о постановке вершин.

### 5.1. Точное решение

В случае изображения ребер в виде ломанных задача определения окончательных координат вершин решается из соображений минимизации числа изломов в проводимых затем ребрах. При этом должен быть учтен и не нарушен порядок следования вершин на каждом из уровней. В такой постановке задача формализуется следующим образом. Пусть у нас есть путь p = (v1, v2, … vk), в котором вершины v2, v3, …, vk-1 являются фиктивными. Если бы такой путь был изображен прямолинейным отрезком, то для i от 2 до k-1 должно выполнятся равенство углов наклона отрезков ломанной

$$y(v_i) - y(v_1) = \frac{1-i}{k-1}(y(v_k) - y(v_1)).$$

Затем, определяя функцию g(p) как

$$g(p) = \sum_{}^{k-1}(y(v_i) - a_i)^2$$
$$a_i = \frac{i-1}{k-1}(y(v_k) - y(v_1)) + y(v_1),$$

мы можем попытаться глобально минимизировать суммарное отклонение от прямолинейности: $\min \sum g(p)$, учитывая ограничение y(w) – y(z) >= q для всех пар вершин, расположенных на одном уровне так, что вершина w расположена выше вершины z. В этом случае q будет минимально возможным вертикальным расстоянием между вершинами.

Функцию для минимизации можно слегка модифицировать, например, так, чтобы отдать предпочтение ортогональным дугам. Для этого надо минимизировать квадрат разности вертикальных координат для всех ребер

$$\sum_{(u,v)\in E}(y(u) - y(v))^2 .$$

Также возможно введение дополнительных весов, которые призваны, чтобы распрямлять именно длинные ребра — как потенциальных обладателей большого количества изломов [40]

$$\sum_{(u,v)\in E}\Omega(u,v)\omega(u,v)\,|\,y(u) - y(v)\,|,$$

где $\omega(u,v)$ — мера важности спрямления ребра (u,v), $\Omega(u,v)$ — коэффициент для спрямления длинных ребер. Авторы подхода предлагают опреде-

лить $\Omega(e)$ следующим образом: $\Omega(e) = 8$, если оба конца ребра являются фиктивными вершинами, $\Omega(e) = 2$, если только одна из конечных вершин фиктивная, и $\Omega(e) = 1$ в остальных случаях.

Здесь стоит отметить два существенных отрицательных момента данного способа определения вертикальных координат вершин. Во-первых, решение данной минимизационной задачи с квадратичным относительно количества вершин набором ограничений может потребовать существенных временных ресурсов. Во-вторых, точное решение с красивыми прямолинейными ребрами может тем не менее оказаться никуда не годным ввиду чрезмерно разросшейся площади изображения. Для избежания разрастания ширины изображения можно усилить налагаемые на вертикальные координаты ограничения, однако это лишь увеличит сложность задачи и возможно элиминирует существование точного решения.

## 5.2. Эвристики

Ввиду перечисленных выше проблем, возникающих при попытках найти точное решение глобальной задачи спрямления ребер, на практике широко применяются эвристические подходы к решению данной проблемы.

Одной из очевидных эвристик является использование идеи барицентров в сочетании с ограничениями на порядок вершин, полученными на предыдущем шаге. Вершины последнего уровня распределяются равномерно на некотором отрезке вертикальной прямой, выделенной под вершины этого уровня. Затем для каждого следующего уровня координаты его вершин последовательно определяются как среднее арифметическое координат их соседей из уже поставленных уровней. При этом, как правило, не допускается нарушение изначального порядка вершин на уровне.

Существует также набор локальных эвристик [40], которые применяются к уже расставленным по вертикали вершинам для того, чтобы улучшить общую картину. Такие локальные эвристики, как правило, применяются последовательно до тех пор, пока не достигнет локального минимума некоторая весовая функция либо не будет совершено определенное количество итераций. В качестве оптимизируемой функции может выступать любая глобальная функция, оценивающая общую длину ребер или их общую прямолинейность.

Хорошие результаты могут быть получены применением методов, основанных на упруго-силовой физической модели [50, 37]. В таких алгоритмах положение вершины определяется силами, действующими на нее со

стороны других вершин через ребра-«пружины» или ребра-«резиновые жгуты». На такую систему накладывается ряд ограничений, для того чтобы предотвратить чрезмерное сближение вершин и сохранение предварительно вычисленного порядка вершин на уровне.

Наиболее обстоятельное и обоснованное изложение силового подхода к определению окончательных координат вершин на уровне сформулировано в [67]. Метод «маятника» интерпретирует вершины как грузы, подвешенные на ребра-струны. Далее такая система приходит в движение под действием гравитационного поля, и вершины, толкая друг друга (но не перескакивая), занимают положение, соответствующее минимуму потенциальной энергии системы. В [67] описан итерационный процесс, позволяющий смоделировать поведение такой системы и найти ее финальное положение.

Стоит отметить, что зачастую алгоритм определения вертикальных координат вершин и проведения ребер очень сильно зависит от особенностей визуализирующей системы. Так, существенным может оказаться форма изображаемых вершин, способ “прикрепления” ребер к вершинам, форма ребер и т.д. В этих случаях возможны существенные модификации алгоритма определения вертикальных координат вершин.

## 6. ПРЕДВАРИТЕЛЬНЫЕ ПРЕОБРАЗОВАНИЯ ГРАФА

Как уже было сказано, данная методология может быть применена не только к ациклическим графам, но также к графам, близким к ациклическим, и даже просто к неориентированным графам. В этих случаях, когда структура графа не соответствует нашему изначальному предположению об ацикличности, требуется произвести некоторые предварительные преобразования над графом. Суть и цель этих преобразований заключается в обратимом преобразовании структуры, так чтобы после размещения ациклического графа возможно было безболезненно для качества получаемого изображения восстановить структуру изначального графа и, тем самым, построить его изображение.

Существует несколько различных подходов к таким преобразованиям [20, 65, 74]. Например, можно предложить склеить каждый из циклов в одну вершину или разместить каждый из циклов на одном уровне. Возможна также вставка в цикл дополнительной вершины, которая будет его разрезать. Однако данная группа преобразований существенно изменяет структуру графа, и это, как правило, сильно сказывается на качестве полу-

чаемого изображения. Это связано с тем, что такие преобразования разрушают имеющуюся иерархическую структуру графа.

Хорошо известна проблема нахождения минимального множества дуг, разрезающих все циклы, т.е. таких дуг, выкидывание которых из графа сделает его ациклическим. Эта задача является NP-полной [38], однако для ее решения существуют хорошие линейные эвристики. К сожалению, данный метод сведения графа к ациклическому не совсем применим в случае иерархического подхода. Проблема заключается в том, что после построения изображения графа, которое не учитывает выкинутые на первом этапе ребра, задача проведения таких ребер становится слабо формализуемой и крайне тяжелой.

Следующий, общепринятый, подход к обратимому преобразованию структуры графа предлагает «разворачивать» часть ребер вместо «выкидывания» их из графа. Преимущество данного способа заключается в том, что развернутые ребра участвуют в построении финального изображения наравне с обычными родными ребрами графа. Для этих ребер также решаются задачи спрямления и минимизации пересечений. После построения конечного изображения у развернутых сначала ребер еще раз меняется ориентация. Здесь встает задача отыскания минимального множества ребер графа, разворот которых сделает его ациклическим. Стоит отметить, что, несмотря на кажущуюся существенной разницу формулировок, эта проблема эквивалентна проблеме отыскания минимального множества дуг, разрезающих все циклы. Эквивалентность предлагает использовать похожие эвристики для решения и этой задачи.

Для упрощения анализа нижеприведенных эвристик будем считать наш граф простым ориентированным графом без мультиребер. Для этого в исходном графе следует заменить мультидуги на ребра с соответсвующим весом. Отдельное внимание стоит уделить проблеме придания ориентации для 2-циклов. Если исходный граф содержит 2-циклы, т.е. пары разнонаправленных ребер $\{(u, v), (v, u)\}$, то предлагается сначала совсем удалить такие ребра, а затем к упрощенному графу применить эвристики для нахождения множества разворачиваемых ребер. После этого ребра, образующие 2-циклы, вставляются в граф и им придается одно и то же направление. В [17] показано, что такое направление всегда можно задать так, чтобы в графе не образовались новые циклы.

Наиболее простой и очевидный способ поиска дуг для разворота заключается в выделении какого-либо остовного дерева в графе и развороте тех дуг, которые не соответствуют направлению, заданному остовным деревом. В наихудшем случае при этом будет развернуто $|E| - |V| - 1$ ребер. Еще

один тривиальный метод предлагает выбрать произвольную нумерацию вершин графа и развернуть те дуги, которые идут от вершин с большими номерами к вершинам с меньшими номерами. Если более половины дуг оказывается выкинутыми, то следует поменять нумерацию на обратную, это гарантирует выкидывание не более |E| / 2 ребер в наихудшем случае.
В работе [40] предложено посчитать, сколько раз каждое ребро входит в цикл и развернуть ребро, получившее наибольшую оценку. Процесс продолжается до образования ациклического графа. К сожалению, авторы не приводят теоретических оценок для данного метода.

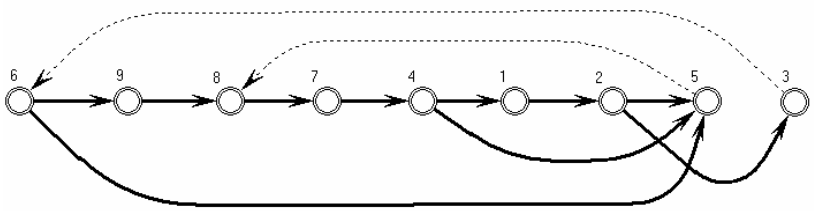


*Рис. 6.* Перенумерация вершин при поиске разворачиваемых ребер

Наиболее же часто используемая эвристика носит название «жадного» алгоритма [30]. Алгоритм гарантирует нахождение ациклического подграфа, в котором не менее чем (*E/2 + V/6*) ребер, и если исходный граф не имеет циклов, то алгоритм оставляет его без изменений. Эвристика предлагает найти такую нумерацию вершин, при которой количество ребер, идущих против этой нумерации, минимально (рис. 6). На каждом шаге алгоритма из исходного графа выкидывается вершина и получает свой номер. Сначала выкидываются все источники и стоки и получают, соответственно, наименьшие и наибольшие номера. В дальнейшем при отсутствии источников и стоков из графа выкидывается вершина с наибольшей разницей количества выходящих и входящих дуг и ей присваивается наименьший из свободных номеров. На каждом шаге алгоритма вместе с удалением вершины удаляются также и все инцидентные ей ребра, таким образом происходит изменение структуры графа. Существует реализация данного алгоритма, имеющая линейную временную сложность. В [29] показано, что для графов, не имеющих вершин со степенью большей 3, верхняя оценка количества разворачиваемых ребер для данного алгоритма есть (*E/3*).

В работе [69] Сандер приводит модификацию “жадного” алгоритма, основанного на другом принципе выбора следующей вершины при отсутствии источников и стоков. Вершина в данном случае выбирается сразу вме-

сте со всей своей сильно связанной компонентой (набором таких вершин $u$, $v$, что одновременно существует путь из $u$ в $v$ и из $v$ в $u$). При надлежащем выборе структур данных такая версия алгоритма будет иметь временную сложность O(EV). Однако, несмотря на впечатляющие практические результаты, теоретическая оценка на количество разворачиваемых ребер так и не была улучшена по сравнению с оригинальной версией алгоритма.

Существуют и другие нелинейные эвристики, которые имеют лучшую оценку работы в наихудшем случае. В [29] представлено обобщение идеи Сандера. Помимо выбора сильно связных компонент, авторы используют тот факт, что для цепочек вершин достаточен разворот всего лишь одного ребра из цепочки. Стягивание цепочек вершин в исходном графе позволяет для графов, не имеющих вершин степени, большей 3, получить хорошую верхнюю оценку на количество разворачиваемых ребер — ($E/4$).

Дальнейшие поиски сложных нелинейных эвристик стоит считать малоперспективными. Во-первых, результат [64] утверждает, что вряд ли есть эвристика с хорошим наихудшим случаем. А, во-вторых, незначительное добавочное уменьшение количества развернутых ребер не оказывает существенного влияние на качество получаемого изображения.

В [14] приведена формулировка данной задачи в терминах линейного целочисленного программирования. Использовние метода ветвей и границ позволяет получить точное решение искомой задачи. Данный метод может быть применен для практического изучения результатов работы других алгоритмов.

## 7. ЗАКЛЮЧЕНИЕ

Отличительной чертой данной методологии является ее интуитивная понятность и масштабируемость. Под масштабируемостью мы здесь понимаем разделение методологии на ряд малопересекающихся задач. Любая финальная реализация — это всегда конструктор, т.е. набор методов, решающих задачи разных этапов. Собирая такой конструктор, исследователь должен выбрать сбалансированный набор методов. Такой набор должен быть непротиворечивым по целям и эквивалентным в терминах временной сложности. Если в набор входит алгоритм, имеющий квадратичную сложность, то нет смысла «экономить» на решении задач других этапов, применяя к ним линейные, но «плохие» эвристики.

Случаи успешного практического применения иерархического подхода к изображению ациклических графов отмечены во многих исследователь-

ских работах [5, 13, 70, 71]. Реализации данного подхода входят во многие системы визуализации графов, в том числе: GraphEd [43], D-ABDUCTOR [73], DAG [41], GD-Workbench [6], Higres [3]. Несмотря на NP-трудность почти всех возникающих подзадач, удается построить реализации, имеющие сложность около квадратичной и получающие приемлемые по качеству изображения.

Практическое сравнение различных подходов к построению размещения ациклических графов можно найти в [11]. Методы, основанные на поуровневом размещении (иерархический подход), хотя и не являются лидерами по всем эстетическим критериям, однако значительно опережают конкурентов на больших графах, возникших в реальных приложениях.

Из существенных нерешенных проблем стоит отметить задачу определения окончательных вертикальных координат для вершин. Размерность задачи линейного программирования делает использование симплекс-метода малопригодным для ее решения. Существующие же эвристики слабо формализуемы и не гарантируют отсутствия даже локальных дефектов у получающегося изображения.

Из теоретических оценок характера и свойств получаемого изображения до сих пор отсутствует какая-либо известная автору оценка площади изображения. Такая оценка до сих пор была приведена лишь для случаев планарного ациклического графа [19, 52].

## СПИСОК ЛИТЕРАТУРЫ

1. **Евстигнеев В. А., Касьянов В.Н.** Базисные алгоритмы обработки бесконтурных графов. — Новосибирск, 1995. — С. 100–112.
2. **Касьянов В.Н.** Применение графов в программировании // Программирование. — 2001. — № 3. — Р.51–70.
3. **Лисицын И. А.** Применение системы HIGRES для визуальной обработки иерархических графовых моделей. // Проблемы систем информатики и программирования. — Новосибирск, 1999. — С. 64–77.
4. **Филаткина Н. Н.** Графовые средства визуализаии свойств программ. — Квалификационная работа на соискание степени бакалавра. — Новосибирск: НГУ, 1998. — 34 с.
5. **Baburin D. E., Bylyonkov M. A., Emelianov P. G., Filatkina N. N.** Visualizing Facilities in Program Reengineering. // Programming and Computer Software. — Interperiodica Publishing, 2001. — Vol. 27, N. 2. — P. 69–77.
6. **Buti L., Di Battista G., Liotta G., Tassinari E., Vargiu F., Vismara L.** GD–Workbench: A system for prototyping and testing graph drawing algorithms // Lect. Notes Comput. Sci. — 1996. — Vol. 1027. — P.111–122.

7. **Brockenauer R., Cornelsen S.** Drawing Clusters and Hierarchies // Lect. Notes Comput. Sci. — 2001. — Vol. 2025. –P. 87–120.
8. **Ball T., Eick S.** Software visualization in large // IEEE Comput. — 1996. — Vol. 29, N.4. — P. 25–39.
9. **Di Battista G., Eades P., Tamassia T., Tollis I.G.** Algorithms for drawing graphs: annotated bibliography // Comput. Geom. Theory Appl. — 1994. — P. 235–282.
10. **Di Battista G., Eades P., Tamassia T., Tollis I.G.** Algorithms for the visualization of graphs. — Prentice–Hall, 1999. — 398 p.
11. **Di Battista G., Garg A., Liotta G., Parise A., et al.** Drawing directed acyclic graphs: an experimental study // Lect. Notes Comput. Sci. — 1997. — Vol. 1190. — P.76–91.
12. **Biedl T., Kant G**. A better heuristic for ortogonal graph drawing // Lect. Notes Comput. Sci. — 1994. — Vol. 855. — P. 124–135.
13. **Di Battista G., Lillo R., Vernacotola F.** Ptolomaeus: the web cartographer // Lect. Notes Comput. Sci. — 1998. — Vol. 1547. — P. 444–445.
14. **Bastert O., Matuszewski C.** Layred drawings of digraphs // Lect. Notes Comput. Sci. — 2001. — Vol. 2025. — P. 87–120.
15. **Bentley J., Ottman T.** Algorithms for reporting and countng geometric intersections // IEEE Trans. Comput. — 1979. — Vol. 28. — P. 643–647.
16. **Bohringer K., Paulisch F. N.** Using constraints to achieve stability in automatic graph layout algorithms // Proc. of the ACM CHI'90. — 1990. — P. 43–51.
17. **Berger B., Shor P.** Approximation algorithms for the maximum acyclic subgraph problem // Proc. of the 1st ACM–SIAM Sympos. Discrete Algorithms. — 1990. — P. 236–243.
18. **Di Battista G., Tamassia R.** Algorithms for plane representations of acyclic digraphs // Theor. Comput. Sci. — 1988. — Vol. 61. — P. 175–198.
19. **Di Battista G., Tamassia R., Tollis I.G.** Area requirement and symmetry display of planar upward drawings // Discrete & Computational Geometry. — 1992. — Vol. 7. — P. 381–401.
20. **Carpano M. J.** Automatic display of hierarchized graphs for computer aided decision analysis // IEEE Trans. Syst., Man, Cybern. — 1980. — Vol.10, N. 11. — P. 705–715.
21. **Catarci T.** The assignment heuristic for crossing reduction // IEEE Trans. Syst., Man, Cybern. — 1995. — Vol. 25, N. 3. — P. 515–521.
22. **Cohen R.F., De Battista G., Tamassia R., Tollis I.G.** Dynamic graph drawings: trees, series–parallel digraphs, and planar st–graphs // SIAM J. Comput. — 1995. — Vol. 24, N. 5. — P. 970–1001.
23. **Chazelle B., Edelsbrunner H.** An Optimal Algorithm for Intersecting Line Segments in the Plane // J. ACM. — 1992. — Vol. 39, N. 1. — P. 1–54.
24. **Coffman E.G., Graham R.L.** Optimal scheduling for two processor systems // Acta Informatica. — 1972. — Vol. 1. — P. 200–213.
25. **Cormen T., Leiserson C., Rivest R.** Intorduction to algorithms // MIT Press. — Cambridge, 1990.

26. **Dresbach S.** A new heuristic layout algorithm for DAGs // Operations Research Proceedings / Ed. by Derigh, Bachem & Drexl. — Springer-Verlag. — 1994. — P. 121–126.
27. **Dobkin D.P., Gasner E.R., Koutsofios E., North S.C.** Implementing a general–purpose edge router // Lect. Notes Comput. Sci. — 1998. — Vol. 1353. — P. 262–271.
28. **Eades P., Feng Q.W., Lin X.** Straight–line drawing algorithms for hierarchical graphs and clustered graphs // Lect. Notes Comput. Sci. — 1997. — Vol. 1190. — P. 113–128.
29. **Eades P., Lin X.** A new heuristic for the feedback arc set problem // Australian J. of Combinatorics. — 1995. — Vol. 12. — P. 15–26.
30. **Eades P., Lin X., Smith W.F.** A fast and effective heuristic for the feedback arc set problem // Information Processing Letter. — 1993. — Vol. 47. — P. 319–323.
31. **Eades P., Kelly D.** Heurisitcs for reducing crossings in 2–layered networks // Ars Combin. — 1986. — Vol. 21. — P. 89–98.
32. **Eades P., McKay B., Wormald N.** On an edge crosiing problem // Proc. of the 9th Australian Comput. Sci. Conf. — 1986. — P. 327–334.
33. **Eades P., Sugiyama K.** How to draw a directed graph // J. of Information Processing. — 1990. — Vol. 13. — P. 424–437.
34. **Eades P., Wormald N.** Edge crossing in drawings of bipartite graphs // Algorithmica. — 1994. — Vol. 11. — P. 379–403.
35. **Eades P., Whitesides S.** Drawing graphs in two layers // Theor. Comput. Sci. — 1994. — Vol. 131. — P. 361–374.
36. **Frick A.** Upper bounds on the number of hidden nodes in Sugiyama's algorithm // Lect. Notes Comput. Sci. — 1997. — Vol. 1190. — P. 169–183.
37. **Fruchterman T., Reingold E.** Graph Drawing by Forcedirected Placement // Software — Practice and Experience. — 1991. — Vol. 21. — P. 1129–1164.
38. **Garey M.R., Johnson D.S.** Computers and interactability: a guide to the theory of NP–completeness / Ed. by W.H. Freeman. — New York, 1979.
39. **Garey M.R., Johnson D.S.** Crossing number is NP–complete // SIAM J. Algebraic Discrete Methods. — 1983. — Vol. 4, N. 3. — P. 312–316.
40. **Gasner E.R., Koutsofios E., North S.C., Vo K.P.** A technique for drawing directed graphs // IEEE Trans. Software Eng. — 1993. — Vol. 19, N. 3. — P. 214–230.
41. **Gasner E.R., North S.C., Vo K.P.** DAG — a program that draws directed graphs // Software — Practice and Experience. — 1988. — Vol. 18, N.1. — P. 1047–1062.
42. **Goldberg A.V., Tarjan R.E.** Finding minimum–cost circulations by successive approximation // Mathematics of Operations Research. — 1990. — Vol. 15, N. 3. — P. 430–466.
43. **Himsolt M.** GraphEd: a graphical platform for the implementation of graph algorithms // Lect. Notes Comput. Sci. — 1994. — Vol. 894. — P. 182–193.
44. **Healy P., Kuusik A.** The vertex–exchange graph: a new concept for multi–level crossing minimization // Lect. Notes Comput. Sci. — 1999. — Vol. 1731. — P. 205–216.

45. **He W., Marriott K.** Constrained graph layout // Lect. Notes Comput. Sci. — 1997. — Vol. 1190. — P. 217–233.
46. **Johnson D.** The NP–completeness column: an ongoing guide // J. on algorithms. — 1982. — Vol. 3, N. 1. — P. 215–218.
47. **Junger M., Lee E.K., Mutzel P., Odenthal T.** A polyhedral approach to the multi–layer crossing minimization problem // Lect. Notes Comput. Sci. — 1998. — Vol. 1353. — P. 13–124.
48. **Junger M., Mutzel P.** Exact and heuristic algorithms for 2–layer straightline crossing minimization // Lect. Notes Comput. Sci. — 1995. — Vol. 1027. — P. 337–348.
49. **Junger M., Mutzel P.** 2–layer straightline crossing minimization: performance of exact and heuristic algorithms // J. on Graph Algorithms and Applications. — 1997. — Vol. 1, N. 1. — P. 1–25.
50. **Kamada T., Kawai S.** An algorithm for drawing general undirected graphs // Information Processing Letters. — 1989. — Vol. 31. — P. 7–15.
51. **Kakoulis K.G., Tollis I.G.** An algorithm for labeling edges of hierarchical drawings // Lect. Notes Comput. Sci. — 1998. — Vol. 1353. — P. 169–180.
52. **Lin X., Eades P.** Area requirements for drawing hierarchically planar graphs // Lect. Notes Comput. Sci. — 1998. — Vol. 1353. — P. 219–229.
53. **Lin X.** Analysis of algorithms for drawing graphs: PhD thesis. — Dept. of Comput. Sci.; Univ. of Queensland, 1992.
54. **Lam S., Sethi R.** Worst case analysis of two scheduling algorithms // SIAM J. on Computing. — 1977. — Vol. 6, N. 3.
55. **Levialdi S.** Visual languages: where we do stand? // Lect. Notes Comput. Sci. — 1999. — Vol. 1779. — P. 145–164.
56. **Makinen E.** Experiments of drawing 2–level hierarchical graphs. — Tempere, 1988. —(Tech. Rep./ Dept. of Computer Science, University of Tampere; N A-1988-1.).
57. **Makinen E.** Experiments on drawing 2–level hierarchical graphs // Intern. J. of Comput. and Mathematics. — 1990. — Vol. 36. — P. 175–181.
58. **Mutzel P.** An alternative method to crossing minimization on hierarchical graphs // Lect. Notes Comput. Sci. — 1997. — Vol. 1190. — P. 318–333.
59. **Masuda S., Nakajima K., Kashiwabara T., Fujisawa T.** Crossing minimization in linear embeddings of graphs // IEEE Trans. on Comput. — 1990. — Vol. 39, N. 1. — P. 124 –127.
60. **Messinger E.M., Rowe L.A., Henry R.H.** A divide–and–conquer algorithm for the automatic layout of large directed graphs // IEEE Trans. on Systems, Man, and Cybernetics. — 1991. — Vol. 21, N. 1. — P. 1–12.
61. **Makinen E., Sieranta M.** Genetic algorithms for drawing bipartite graphs // Proc. of the 24th Annual ACM Symposium on the Theory of Computing, STOC'92. — 1994. — P. 527–538.
62. **Matuszewski C., Schonfeld R., Molitor P.** Using sifting for k–layer straightline crossing minimization // Lect. Notes Comput. Sci. — 1999. — Vol. 1731. — P. 205–216.

63. **North S.C.** Incremental layout in DynaDAG // Lect. Notes Comput. Sci. — 1995. — Vol. 1027. — P. 409–418.
64. **Papadimitriou C., Yannakakis M.** Optimization, approximization and complexity classes // J. of Computer and System Sciences. — 1991. — Vol. 43. — P. 425–440.
65. **Robbins G.** The ISI grapher, a portable tool for displaying graphs pictorially. — Los Angeles, 1987.— (Tech. Rep. / Information Sci. Inst., Univ. of Southern California; N ISI/RS-87-196).
66. **Reingold E., Tilford J.** Tidier drawing of trees // IEEE Trans. Soft. Eng. — 1981. — Vol. 7, N. 2. — P. 233–228.
67. **Sander G.** Graph layout through the VCG tool // Lect. Notes Comput. Sci. — 1995. — Vol. 894. — P. 194–205.
68. **Sander G.** A fast heuristic for hierarchical mangattan layout // Lect. Notes Comput. Sci. — 1995. — Vol. 1027. — P. 447– 458.
69. **Sander G.** Graph layout for applications in compiler construction. 1996 — (Tech. Rep./ Universitas des Saarlandes; N A/01/96).
70. **Seemann J.** Extending the Sugiyama algorithm for drawing UML class diagrams: towards automatic layout of object–oriented software diagrams // Lect. Notes Comput. Sci. — 1998. — Vol. 1353. — P. 415–424.
71. **Sander G., Alt M., Ferdinand C., Wilhelm R.** ClaX —a visualized compiler // Lect. Notes Comput. Sci. — 1995. — Vol. 1027. — P. 459–462.
72. **Shieh F.S., McCreary C.L.** Directed graphs drawing by clan–based decomposition // Lect. Notes Comput. Sci. — 1995. — Vol. 1027. — P. 472–482.
73. **Sugiyama K., Missue K.** A generic compound graph visualizer/manipulator: D–ABDUCTOR // Lect. Notes Comput. Sci. — 1995. — Vol. 1027. — P. 500–503.
74. **Sugiyama K., Tagawa S., Toda M.** Methods for visual undertanding of hierarchical systems // IEEE Trans. Syst., Man,and Cybern. — 1981. — Vol. 11, N. 2. — P. 109–125.
75. **Tomii N., Kambayashi Y., Shuzo Y.** On planarization algorithms of 2–level graphs // Papers of Tech. Group on Electronic computers, IECEJ. — 1977. — Vol. 38. — P. 1–12.
76. **Valls V., Marti R., Lino P.** A branch and bound algorithm for minimizing the number of crossing arcs in bipartite graphs // J. of Operational Research. — 1996. — Vol. 90. — P. 303–319.
77. **Warfield J.** Crossing theory and hierarchy mapping // IEEE Trans. Syst., Man,and Cybern. — 1977. — Vol. 7, N. 7. — P. 502–523.
78. **Waddle V., Malhotra A.** An E logE line crossing algorithm for levelled graphs // Lect. Notes Comput. Sci. — 1999. — Vol. 1731. — P. 205–216.