# Циркулярная укладка атрибутированного иерархического графа с портами

В. Н. Касьянов, email: kvn@iis.nsk.su
А. М. Меркулов, email: arseniy.merkulov@gmail.com
Т. А. Золотухин, email: tzolotuhin@gmail.com

Институт систем информатики им. А.П. Ершова СО РАН

***Аннотация.*** *Визуализация информации на основе графовых моделей является ключевым компонентом инструментов поддержки многих приложений в науке и технике. Система Visual Graph предназначена для визуализации больших объемов сложной информации на основе атрибутированных иерархических графовых моделей. В данном докладе представлен алгоритм циркулярной укладки атрибутированных иерархических графов с портами и его эффективная реализация в системе Visual Graph.*

***Ключевые слова:*** *атрибутированные иерархические графы с портами, визуализация информации, система Visual Graph, циркулярная укладка.*

## Введение

Графы широко используются для описания и структурирования информации в предметных областях, где необходимо моделировать связи между объектами. Методы визуализации графов, которые встраивают графы на двух- или трехмерную поверхность, где вершины представлены точками, прямоугольниками или эллипсами, а дуги представлены кривыми, соединяющими их визуализацию, имеют множество областей применения [1, 2, 7, 9].

Для визуального анализа данных и связей между ними используются различные их графовые представления, которые могут сильно различаться в зависимости от области применения и с учетом множества так называемых эстетических критериев. Например, размер изображения или его симметричность часто важны при визуализации графов во многих приложениях. Обычно важно иметь как можно меньшее количество пересечений дуг графа. Однако задача построения изображения, удовлетворяющих всем желаемым критериям, как правило, весьма трудоёмка и даже неосуществима, поскольку обычно критерии приложений по своей сути конфликтуют, и поэтому обычно используются некоторые компромиссы [1].

В некоторых прикладных областях организация информации слишком сложна для моделирования классическими графами, поэтому были введены более мощные формализмы графов и новые методы иерархической визуализации [8, 11, 14]. Практический и общий формализм графов, называемый иерархическими графами и моделями графов, может обрабатывать многие приложения со сложной информацией и поддается рисованию графов [3, 4, 10]. Он формирует основу для естественных методов «абстракции» и «редукции», которые могут применяться исследователями для уменьшения визуальной сложности большого графа. Он делит граф на части (так называемые фрагментами), так что все элементы каждого фрагмента в некотором смысле похожи, и все рассматриваемые фрагменты образуют иерархию вложенных частей графа. Эта иерархия обеспечивает хорошее средство обработки сложности размера для тех приложений, которые обязаны иметь дело с большими объемами данных. Система визуализации Visual Graph, созданная в Институте систем информатики им. А.П. Ершова СО РАН, основана на атрибутивных иерархических графовых моделях и позволяет исследовать сложные структурированные большие данные через их визуальные представления [10].

Во многих приложениях объекты, моделируемые вершинами графа, содержат непересекающиеся логические местоположения (так называемые порты [6]), через которые они (объекты) находятся во взаимосвязи, моделируемой дугами. Например, в графе программы, моделирующем поток данных в программе, операторы программы представляются вершинами графа, операнды операторов (их аргументы и результаты) моделируются портами вершин, а поток данных между результатами и аргументами операторов представлен дугами, соединяющим соответствующие порты [3, 4].

Наглядность полученного изображения графа сильно зависит от того, как его элементы (вершины и дуги) расположены на плоскости. Циклическое изображение графа — это такая укладка графа на плоскости, при которой все вершины графа помещаются на окружность некоторого круга, а каждая дуга рисуется внутри этого круга обычно в виде прямой линии [13]. Однако проблема построения циклического изображения с минимальным количеством пересечением дуг является NP-полной [12]. Циклическая укладка находит свое применение в тех приложениях, где объекты, моделируемые вершинами графа, имеют равный приоритет, и ни один из них не занимает привилегированное положение. Циклические изображения графов используются для визуализации топологий кольцевых и звездных сетей, биологических и социальных сетей, а также небольших кластеров в больших графах.

Поскольку эти приложения работают с большими данными, проблема разработки алгоритма циклической укладки для общих иерархических графов очень актуальна, но в настоящее время только для кластерных графов, представляющих собой простые иерархические графы без портов, был разработан алгоритм циклической укладки [5].

## 1. Иерархические графы с портами и их изображения

Введем некоторые обозначения из [11]. Пусть G — граф некоторого типа, например, G может быть неориентированным или ориентированным графом. Граф G определяется двумя конечными множествами V и E, где элементы V — вершины графа G, а элементы E — дуги графа G. G — тривиальный граф, если $|V| = 1$ и $|E| = 0$.

Граф C называется фрагментом графа G и обозначается $C \subseteq G$, если C включает только элементы (вершины и дуги) графа G. Набор фрагментов F называется иерархией вложенных фрагментов графа G, если следующие два выполнены следующие свойства:

(1) F содержит граф G и все вершины V, и

(2) $C_1 \subseteq C_2$, $C_2 \subseteq C_1$ или $C_1 \cap C_2 = \varnothing$ для любых $C_1, C_2 \in F$.

Иерархический граф H = (G, T) состоит из графа G и корневого дерева T, которое представляет отношение непосредственного включения между элементами иерархии F вложенных фрагментов G. G называется базовым графом H. T называется дерево включения H.

Иерархический граф H называется связным, если каждый фрагмент H является связным графом, и простым, если все фрагменты H являются порожденными подграфами графа G.

Следует отметить, что любой кластерный граф [2] можно рассматривать как простой иерархический граф H = (G, T) такой, что G является неориентированным графом, а любой фрагмент $C \in F$ рассматривается как множество вершин графа C.

Пусть P — конечный набор портов. Порт $p \in P$ — это специальная вершина иерархического графа, которая прикреплена к фрагменту $C \in F$ и служит для обозначения точки, где дуга входит в соответствующий фрагмент C. Предполагается, что каждая дуга иерархического графа с портами является дуга между двумя портами.

Пусть P(C) — упорядоченный набор портов фрагмента C, а $P_i$ (C) — порт C с номером i. Фрагмент C содержит порт p, если p — порт некоторого фрагмента, вложенного в C. Дуга $u = (P_i(w), P_j(z))$ разрезает фрагмент C (u — разрезающая дуга), если C содержит один из элементов множества {w, z} и не содержит другой элемент.

Иерархический граф H с портами является структурным, если у него нет разрезающих дуг.

Пусть дуга u = ($P_i$(w), $P_j$(z)) разрезает фрагмент C. Преобразование удаления из H разрезающей дуги u определяется как замена в H дуги u = ($P_i$(w), $P_j$(z)) двумя дугами ($P_i$(w), $P_t$(z)) и($P_t$(w), $P_j$(z)), где $P_t$(C) — новый фиктивный порт, добавленный к фрагменту C. $H_1$ является структурным расширением иерархического графа $H_2$, если $H_1$ является структурным иерархическим графом, который может быть построен из $H_2$ путем итеративного применения преобразования удаления разрезающих дуг.

Изображение (или укладка) D иерархического графа H является таким представлением элементов H на плоскости, что выполняются следующие свойства.

1. Каждый порт, вершина и фрагмент графа G представлен простой замкнутой областью (в нашем случае - циклом). Область определяется ее границей (простой замкнутой кривой на плоскости), которая делит плоскость на две части: внутреннюю грань и внешнюю грань.

2. Для любых $C_1$, $C_2 \in F$ пересечение областей $C_1$ и $C_2$ пусто тогда и только тогда, когда $C_1 \cap C_2 = \varnothing$, а область каждого фрагмента $C \in F$ включает в себя области всех вложенных в него фрагментов.

3. Любые два порта каждого фрагмента $C \in F$ представлены двумя непересекающимися областями. Область каждого порта C разделена на три непустых набора точек: точки внутренней грани области C, точки ее границы и точки ее внешней грани.

4. Каждая дуга H представлена простой кривой между изображениеми ее конечных точек (портов).

5. Все дуги любого фрагмента C фрагмента H расположены внутри области C.

Изображение D иерархического графа H является циркулярной укладкой, если для любого фрагмента C из F все его сыновья в T (все фрагменты из F, непосредственно вложенные в C) помещаются на окружности некоторого круга.

## 2. Алгоритм циркулярной укладки

Ниже предполагается, что на вход алгоритма подается некоторый иерархический граф H = (G, T) с портами, а выходом алгоритма является циркулярная укладка D структурного расширения графа H.

Работа алгоритм осуществляется в процессе обхода дерева вложенности T в глубину. На каждом шаге происходит укладка некоторого фрагмента с использованием размеров и расположений непосредственно вложенных в него вершин или других фрагментов. Отметим, что после построения изображения некоторого фрагмента можно вычислить его размеры и использовать их при укладке фрагмента, в который данный фрагмент непосредственно вложен.

Само построение изображения одного фрагмента состоит из трех этапов: вычисление размеров вершин, упорядочивание вершин относительно друг друга, укладка дуг.

Для построения циркулярного изображения предлагается разделить алгоритм на три прохода по всем уровням иерархии. Это обусловлено тем, что вычисление размеров и укладку дуг нужно производить в разном порядке: вычисление размеров нужно начинать с листьев дерева вложенности, а укладку дуг с корня.

Рассмотрим этап вычисления размеров вершин. Для всех тривиальных фрагментов (или листьев T) размер уже задан на входе в алгоритм. Для других фрагментов их размер подсчитывается на основе размеров вершин и фрагментов, лежащих внутри. Из этого следует, что первый этап нужно производить от низкого уровня иерархии вверх по уровням. На данном этапе выполняется следующие шаги:

1. Вычисляется радиус окружности, на которой будут расположены вершины. Это можно сделать, т. к. размеры укладываемых вершин уже известны. Для этого шага потребуется вычислить минимальный радиус окружности, которая бы смогла покрыть собой любую из заданных вершин.

2. Все вершины равномерно распределяются по окружности вычисленного радиуса.

3. Если заданные вершины являются внутренними для некоего фрагмента, то считается размер данного фрагмента. После чего равномерно по окружности фрагмента размещаются реальные порты, если они есть.

Если рассматривать фиктивные порты, то в них снаружи обязательно приходит дуга, поскольку это является условием существования фиктивного порта. В случае реальных портов это не так, они могут соединяться только с внутренними вершинами фрагмента. Соответственно, требуется задать позиции для портов, для которых внешних дуг нет. Но определение внешних связей — довольно дорогостоящая операция. Поэтому позиции задаются всем портам, а на этапе укладки дуг некоторые из них меняются местами.

Задачей второго этапа является нахождение порядка вершин на окружности с целью минимизации количества пересечений дуг.

Следует отметить, что количество пересечений дуг в циркулярном представлении графа не зависит от конечных координат вершин, а зависит только от их относительного положения на окружности (их порядкового номера). Таким образом, данная задача является комбинаторной, а не геометрической. Однако эта задача является NP-полной [12]. Поэтому ниже мы приведем две эвристики, которые

определяют порядок расположения фрагментов за полиномиальное время, но не гарантируют лучший результат на любом графе. Обозначим их как эвристики перестановки и группировки, исходя из особенностей каждой из них.

Эвристика группировки основана на предположении, что количество пересечений между дугами будет уменьшаться, если наиболее связанные дугами фрагменты будут размещены рядом. Такое расположение фрагментов приводит к уменьшению количества дуг, лежащих близко к центру окружности. Как правило, такие дуги вызывают большое количество пересечений, так как их длина близка к максимально возможной при данном расположении. Эвристика группировки состоит из следующих шести шагов.

1. Пусть S — множество всех рассматриваемых фрагментов.

2. Пусть C — наиболее связный с другими фрагмент из S.

3. Выбирается группа всех тех фрагментов, которые соединены дугами с C.

4. Все фрагменты данной группы удаляются из S.

5. Фрагменты удаленной группы сортируются по количеству дуг между ними и размещаются так, чтобы наименее соединенные фрагменты располагались «снаружи».

6. Если множество S не пусто, алгоритм продолжается с шага 2.

Следует отметить, что использование такой эвристики приводит к увеличению количества дуг, лежащих на границе окружности укладки.

Теперь разберем эвристику перестановок. Алгоритм данной эвристики использует циклическую перестановку соседних вершин для минимизации пересечений. Формализуем шаги данной эвристики в виде следующей последовательности шагов:

1. Пусть S — последовательность всех рассматриваемых фрагментов.

2. Алгоритм шаг за шагом рассматривает все смежные пары фрагментов из S, начиная с пары, состоящей из первого и второго фрагментов, и выполняет шаг 3 для каждой из них. Последней рассматривается пара, состоящая из последнего и первого фрагмента S.

3. Рассматриваются инцидентные дуги для текущей пары фрагментов. Подсчитывается количество их пересечений с другими дугами. Если перестановка фрагментов в рассматриваемой паре приводит к уменьшению числа пересечений, то фрагменты меняются местами.

4. Если поменялись местами фрагменты какой-то пары, то алгоритм продолжается с шага 2.

Данный алгоритм неэффективен по времени, но во многих случаях редуцирует пересечения настолько, насколько это возможно. Он подходит для применения к малым графам, где недостаток скорости не заметен.

Рассмотрим третий этап алгоритма, связанный с укладкой дуг. Дуги задаются как ломанные, проходящие через опорные точки. На данном этапе для каждой дуги вычисляется набор этих точек. Для каждой дуги гарантированно существуют две точки — начало и конец. Их можно вычислить, исходя из взаимного положения инцидентных вершин и угла между дугой и осью абсцисс. Если дуга связана с портом, находящимся на более низком уровне, то также определяется его позиция. Процесс укладки идет от высокого уровня иерархии к низкому, т. к. чтобы уложить дуги, нужно знать координаты портов, которые определяются исходя из дуг более высокого уровня. На данном этапе должны быть выполнены следующие шаги:

1. На входе имеется пустой список, в который будут добавляться опорные точки.
2. Вычисляется точка «начало дуги» и добавляется в начало списка.
3. Если существует точка, в которой дуга приближается к не инцидентной вершине на заданный радиус, то в список начинают добавляться точки, проектирующие дугу на окружность радиуса с центром в данной вершине.
4. Проектирование заканчивается, если дуга вышла из окружности.
5. Вычисляется точка «конец дуги» и добавляется в конец списка.
6. Если у вершины, из которой исходит дуга, имеется соответствующий порт, то ему назначается позиция «начало дуги».
7. Аналогично, если у вершины, в которую приходит дуга, есть соответствующий порт, то ему назначается позиция «конец дуги».

При укладке дуг может возникнуть проблема пересечения дугой вершины, шаги 3 и 4. Для решения данной проблемы были выбраны округлые вставки в дуги. Дуги задаются набором опорных точек, через которые они должны проходить. Если дуга приближается к не инцидентной вершине на заданное расстояние, то в этом месте начинают добавляться опорные точки, проектирующие дугу на окружность с центром в данной вершине и радиусом. Эти точки добавляются до тех пор, пока дуга не выйдет из окружности. Данное решение не увеличивает площадь графа. Для графов с большим числом пересечений это не приводит к эстетическому улучшению изображения, т. к. общее расположение дуг хаотично и локальные улучшения не заметны. Кроме того, это требует дополнительных временных затрат.

Таким образом, для достаточно больших графов из алгоритма укладки дуг нужно убрать шаги 3 и 4.

## 3. Реализация алгоритма

Представленный алгоритм был реализован как расширение системы Visual Graph [10]. Для лучшего понимания необходимо сказать несколько слов об этой системе. Visual Graph — это универсальная система визуализации атрибутированных иерархических графов, которая распространяется по лицензии BSD и предназначена для визуализации на основе графовых моделей большого размера сложной структурированной информации в компиляторах и других системах конструирования программ. Номенклатура графовых моделей, используемых в системах конструирования программ, может отличаться от одной системы к другой. Более того, представления одних и тех же графовых моделей могут различаться между собой в разных системах и даже в разных версиях одной и той же системы. Однако все эти теоретико-графические модели являются частными случаями иерархических графов с атрибутами. Поэтому предполагается следующий сценарий использования системы Visual Graph. Система Visual Graph считывает представленную системой конструирования графовую модель из файла в одном из поддерживаемых ею форматов (GraphML, DOT или GML), а затем предоставляет пользователю инструменты навигации по графовой модели, а также алгоритмы для визуализации разных её частей.

В системе Visual Graph присутствует механизм для добавления расширений, с помощью которого был добавлен циркулярный укладчик, а также отображены настройки укладчика через визуальные компоненты.

Таблица 1

*Сравнение количества пересечений дуг*

| Граф | Эвристика группировки | yEd |
|---|---|---|
| 8 вершин, 6 дуг | 0 | 0 |
| 10 вершин, 10 дуг | 5 | 3 |
| 14 вершин, 22 дуги | 33 | 20 |
| 40 вершин, 39 дуг | 1 | 13 |

Некоторые результаты по сравнению числа пересечений при применении к небольшим простым графам нашего эвристического алгоритма группировки с применением к ним алгоритма циклической укладки системой yEd [5] представлены в таблице 1. Как видно из

полученных результатов, нет четкого превосходство одного алгоритма над другим, и для визуализации различных простых графов необходимо выбирать алгоритм исходя из их индивидуальных характеристик.

Также были проведены замеры временных затрат алгоритма на нескольких больших графах. Их результаты приведены в таблице 2.

Таблица 2

*Измерение временных затрат*

| Граф | Расширение |
|---|---|
| 1100 вершин, 900 дуг | 2 с |
| 3000 вершин, 2000 дуг | 25 с |
| 8044 вершины, 10648 дуг | 1 мин 48 с |

## Заключение

В докладе описывается созданный алгоритм циклической укладки, который строит изображения на плоскости атрибутированных иерархических графов с портами за квадратичное время, а также его эффективная реализация в рамках системы Visual Graph. Алгоритм учитывает критерий минимизации пересечений дуг и использует закругленные дуговые вставки для решения задачи пересечения дуги с вершиной. Его способность рисовать графы с портами очень важна для многих приложений и является новой для алгоритмов циклической укладки графа на плоскости.

В будущем алгоритм может быть улучшен по следующим направлениям. Во-первых, мы планируем улучшить эвристику перестановки. Например, это можно сделать, добавив перестановки по соседним уровням или улучшив методы минимизации пересечений. Во-вторых, мы планируем учесть и сохранить исходный порядок портов фрагментов. Мы также планируем уменьшить площадь изображения графа за счет более экономного распределения фрагментов иерархических графов по окружностям, а также расширить алгоритм, добавив компонент, ориентированный на рисование ациклических фрагментов.

## Список литературы

1. Касьянов, В. Н. Визуализация графов и графовых моделей / В. Н. Касьянов, Е. В. Касьянова. — Новосибирск: ООО «Сибирское Научное Издательство», 2010.

2. Касьянов, В. Н. Графы в программировании: обработка, визуализация и применение / В. Н. Касьянов, В. А. Евстигнеев. — СПб.: БХВ-Петербург, 2003.

3. Касьянов, В. Н. Методы и алгоритмы визуализации графовых представлений функциональных программ / В. Н. Касьянов, Т. А. Золотухин, Д. С. Гордеев // Программирование. — 2019. — № 4. — с. 19–27.

4. Система облачного параллельного программирования CPPS: визуализация и верификация Cloud Sisal программ / В. Н. Касьянов, Д. С. Гордеев, Т. А. Золотухин [и др.]; под ред. В. Н. Касьянова. — Новосибирск: ИПЦ НГУ, 2020.

5. Система yEd [Электронный ресурс] — Режим доступа: http://www.yworks.com

6. Формат GraphML [Электронный ресурс] — Режим доступа: http://graphml.graphdrawing.org

7. Di Battista, G. Graph Drawing: Algorithms for Visualization of Graphs / G. Di Battista, P. Eades [et all]. — Prentice Hall, 1999.

8. Feng, Q. W. Planarity for clustered graphs / Q. W. Feng, R .F. Cohen, P. Eades // Lecture Notes in Computer Science. — 1995. — Vol. 979. — pp. 213–226.

9. Herman, I. Graph visualization and navigation in information visualization: a survey / I. Herman, G. Melançon, M. S. Marshall // IEEE Trans. on Visualization and Computer Graphics. — 2000. — Vol. 6. —pp. 24–43

10. Kasyanov, V. N. A system for visualization of big attributed hierarchical graphs / V. N. Kasyanov, T. A. Zolotuhin // Intern. Journal of Computer Networks & Communications. — 2018. — Vol. 10, № 2. —pp. 55 – 67.

11. Kasyanov, V. N. Information visualization based on graph models / V. N. Kasyanov, E. V. Kasyanova // Enterprise Information Systems — 2013. — Vol. 7, № 2. — pp. 187–197.

12. Masuda, S. On the NP-completeness of a computer network layout problem / S. Masuda, T. Kashiwabara [et all] // Proc. IEEE 1987 International Symposium on Circuits and Systems. — Philadelphia: PA, 1987. — pp. 292–295.

13. Six, J. M. A framework for circular drawings of networks / J. M. Six, I. G. Tollis // Lecture Notes in Computer Science. — 1995. — Vol.1731. —pp. 107–116.

14. Sugiyama, K. Visualization of structured digraphs / K. Sugiyama, K. Misue // IEEE Trans. on Systems, Man and Cybernetics. — 1999. — Vol. 21, № 4. —pp. 876–892.