Интерпретационный метод визуализации графовых алгоритмов

Гордеев Дмитрий Станиславович, gds@iis.nsk.su, ИСИ СО РАН

Аннотация

В статье описывается подход к визуализации графовых алгоритмов, позволяющий задание алгоритма в качестве параметра. Основными преимуществами данного подхода являются возможность задания алгоритма в текстовой форме в качестве параметра и гибко настраивать результирующую визуализацию. Визуализация алгоритмов осуществляется с помощью множества настраиваемых эффектов. В качестве входных графов рассматривается класс иерархических графов. Использование иерархических графов позволяет облегчить визуализацию дополнительных структур данных, так как графовое представление структуры данных можно включить в иерархию графовой модели. Описываемый подход может использоваться как для создания образовательных систем, так и для исследования графовых алгоритмов.

Ключевые слова: *граф, иерархический граф, алгоритм, визуализация, визуальный эффект.*

1. ВВЕДЕНИЕ

В настоящее время методы визуализации графов имеют множество применений. Графы и графовые модели используются для задач, оперирующих информацией, представимой в виде объектов и связей между ними. Например, синтаксические деревья в трансляторах, раскраска графов в конструировании электрических схем, поиск кратчайшего пути в компьютерных играх или геоинформационных системах и т.д [1]. Целью рисования графов является получение изображений графов, простых для рассмотрения и понимания. Рисование графов используется во многих приложениях для проектирования и анализа коммуникационных сетей, связанных документов, а также статических и динамических структур программ. Часто подобные системы подвергаются внутренним или внешним воздействиям, например, изменяющим связи между объектами. В случае, если такие процессы можно формализовать и представить в виде некоторого алгоритма возникает необходимость построить графическое представление процесса. Существуют методы позволяющие представлять такие алгоритмы, как в динамической форме, так и в статической форме. К статическим формам можно отнести блок-схемы. К динамическим формам можно отнести методы, генерирующие последовательности рисунков, изображающих промежуточные состояния графовых моделей. На приведенном ниже рисунке представлен пример динамической системы визуализации алгоритмов [4,6].

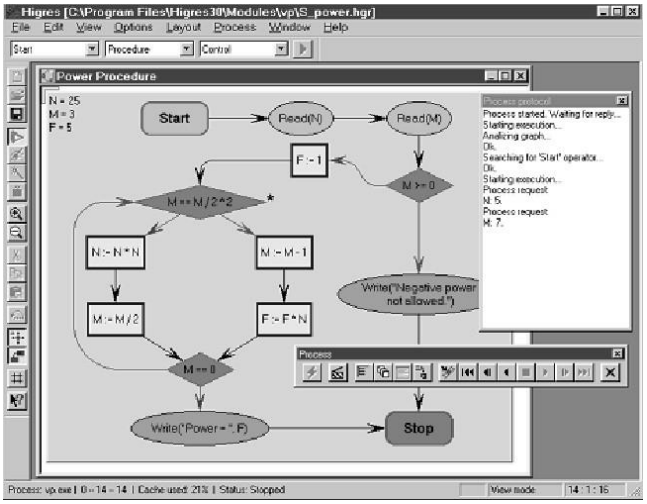

Рис. 1: Выполнение алгоритма, реализованного в виде внешнего модуля к системе Higres.

Существование подобных методов позволяет изучать графовые алгоритмы как в целом, так и в приложении к связанным системам. Большинство работ по визуализации алгоритмов сконцентрировано на построении примеров визуализации [5]. Основной характерной особенностью таких работ можно назвать узкую специализацию, в том смысле, что для каждого нового алгоритма требуется создавать новый визуализатор.

2. МЕТОДЫ ВИЗУАЛИЗАЦИИ

В данной работе описывается метод построения визуализаций основанный на визуальных эффектах, генерируемых по входному алгоритму. Обычно элементам графа сопоставляются некоторые графические примитивы, что дает возможность построить его изображение. Например, вершины отображаются окружностями, а дуги прямыми, ломаными или гладкими кривыми. Среди применяющихся методов визуализации алгоритмов можно выделить два метода: метод интересующих событий и метод, основанный на данных [2]. Первый метод основан на выделении событий, которые происходят во время исполнения алгоритма. Например, сравнение величин атрибутов некоторых элементов графа, добавление вершины или удаление дуги. Суть этого метода в том, чтобы для каждого такого события реализовать визуальный эффект. Второй метод, основан на изменение данных. Во время работы изменяется состояние памяти, например, значения переменных. Далее эти изменения некоторым образом визуализируются. В простейшем случае для этого используется отображение значений переменных в таблице. Такой метод применяется в отладчиках интегрированных систем разработки программного обеспечения.

У существующих визуализаторов алгоритмов есть ряд недостатков. Если есть необходимость построить визуализацию алгоритма, текстовое представление которого

отличается от текста оригинального алгоритма, потребуется заново изготовить визуализатор. Также визуализаторы обычно не отображают соответствие между инструкциями алгоритма и генерируемыми визуальными эффектами. Часто визуализаторы не позволяют перенастроить визуальные эффекты, соответствующие событиям. К недостаткам также можно отнести излишнюю перегруженность исходного текста алгоритма декларативными инструкциями. На рисунке 2 представлен кадр, полученный при визуализации алгоритма в системе Leonardo [3, 7]. Как видно на рисунке, в системе Leonardo используются директивы в специальном формате, /** Not VisualUpdate **/. Такие декларативные конструкции используются для группировки визуальных событий или же для непосредственного запуска визуальных эффектов.

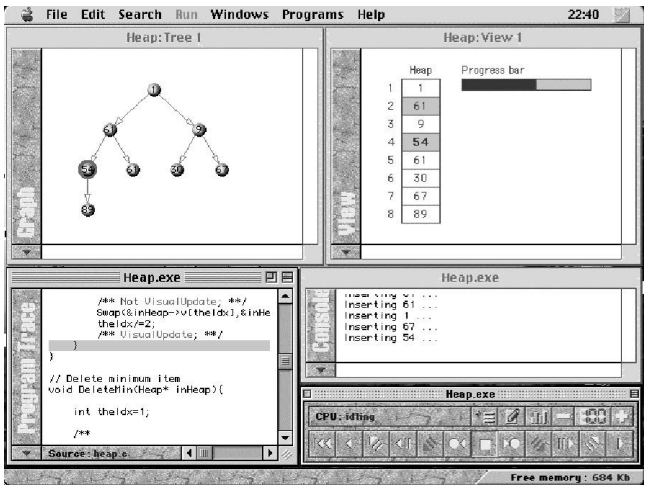

Рис. 2: Пример визуализации операций над binary heaps в системе Leonardo, A C Programming Environment for Reversible Execution and Software Visualization.

3. МОДЕЛЬ ИНТЕРАКТИВНОЙ ВИЗУАЛИЗАЦИИ

Для решения этих задач предложена модель визуализации графовых алгоритмов основанная на динамическом подходе. Суть данного подхода заключается в том, что заданный алгоритм формулируется на языке программирования, допускающем использование графовых структур данных, а также с возможностью исполнить программу, полученную из текста алгоритма после предварительной обработки. Во время исполнения полученной программы генерируется информация, которая используется для визуализации оригинального алгоритма. Примером конструкции, использующей графовые структуры данных, является операция добавления дуги или изменение атрибута вершины графа. В приведенном ниже примере представлен алгоритм обхода в ширину для неориентированного графа. В данном случае используются конструкции Get(идентификатор вершины, имя атрибута) и Set(идентификатор вершины, имя атрибута, значение) для чтения и изменения значения атрибута вершины соответственно. Для построения визуализации алгоритма обхода в ширину вершинам входного графа добавляется атрибут state, по значению которого можно определить была ли посещена вершина в процессе обхода графа или нет.

```
VertexQueue.Enqueue(1, Graph.Vertices[0]);
while (WhileCondition(2, VertexQueue.Count > 0))
{
  Vertex v = VertexQueue.Dequeue(4);
  Set(5, v, "state" , "visited");
  foreach(Vertex n in ForeachCollection(6, v.Incidents))
  {
    string c = Get(8, n, "state");
    if(IfCondition(9, c != "visited"))
    {
      VertexQueue.Enqueue(11, st);
    }
  }
}
```

Для решения задачи визуального выделения текущей исполняемой инструкции в кадре, сгенерированном во время исполнения данной инструкции, используется следующий подход. Оригинальный текст алгоритма преобразуется так, чтобы добавить в каждую значимую строку текста номер этой строки. В представленном выше примере добавленные конструкции выделены жирным шрифтом. Во время исполнения преобразованной программы каждая инструкция алгоритма генерирует информацию, описывающую номер исполняемой строки, идентификатор обрабатываемого элемента графа, название атрибута графового элемента, предыдущее значение атрибута, новое значение атрибута, а также временную метку.

Данная информация о каждой исполненной инструкции позволяет получить историю исполняемых операций, содержащий подробную информацию о состоянии графовой модели во время исполнения алгоритма. Далее полученная история операций и исходный граф используется для построения визуализации алгоритма. Каждой записи истории соответствует некоторый графический эффект. Простейший пример такого графического эффекта это цветовое выделение исполняемой строки алгоритма.

4. СИСТЕМА ВИЗУАЛИЗАЦИИ АЛГОРИТМОВ

С использованием предложенной модели реализован прототип системы визуализации алгоритмов. Система визуализации графовых алгоритмов содержит несколько компонентов. Это модуль исполнения алгоритма, графовый редактор и визуализатор графовых алгоритмов. Назначением модуля исполнения является запуск программы, полученной из текста алгоритма-параметра и порождение истории исполненный операций. Таким образом, следует заметить, что исполнение алгоритма-параметра отделено от визуализации. Это позволяет исполнить алгоритм однажды, и затем строить и уточнять визуализацию без повторных запусков. Это может быть полезно при визуализации вычислительно-емких алгоритмов, когда повторное исполнение алгоритма требует существенного времени.

Чтобы обеспечить корректное функционирование модуля исполнения, требуется выполнение важных условий. Во-первых, для реализации данного модуля подходит любой существующий компилятор или интерпретатор. Соответственно, входной алгоритм должен быть сформулирован с использованием языка выбранного компилятора или интерпретатора. Этот пункт не делает существенных ограничения на язык описания входных

алгоритмов, так как многие языки программирования допускают использование графовых конструкций. Во-вторых, запрещается использование специальных инструкций более одного раза в одной строке. Нарушение этого условия приводит к генерации нескольких визуальных эффектов при том, что выделяться цветом будет только одна строка из текста алгоритма.

Вторым основным компонентом является визуализатор. Он получает на вход текст алгоритма, история операций, граф и дополнительные графические настройки. Каждая запись истории операций содержит информацию об исполняемой инструкции алгоритма. Специальные инструкции являются функциями: Set(…), Get(…), IfCondition(…), WhileCondition(…) and ForeachCollection(…). Первый аргумент этих функций является номером строки в тексте алгоритма. Функции IfCondition(…) and WhileCondition(…) не производят никаких изменений в состоянии графовой модели, но позволяет генерировать эффект выделения строки текста с условным оператором if и оператором цикла while. ForeachCollection(…) инструкция используется для генерации визуального выделения множества вершин перед тем, когда множество будет обрабатываться в цикле foreach. Чтобы вставить эти функции в подходящие места текста алгоритма достаточно использовать контекстную замену.

Каждая запись истории может содержать информацию о текущем значении атрибута для некоторого элемента графа, будь то вершина, ребро или порт. Для вершины, инцидентной некоторой дуге, порт это точка, где дуга входит в вершину. Выделение порта в отдельную сущность обусловлено необходимостью изображать точки входа и выхода дуг из вершин. Часто, если из вершины выходит несколько дуг, необходимо построить изображение, в кототором точки выхода необходимо различать. Выделенная сущность порта упрощает вычисление координат графических примитивов, представляющих дуги графа. При этом встает задача определения связи между информацией записи истории и соответсвующим визуальным эффектом. В тексте алгоритма допускается использование произвольных имен атрибутов. В этом случае требуется вмешательство пользователя, чтобы установить интерпретацию именами атрибутов с помощью множества доступных визуальными эффектами.

На представленном ниже рисунке представлен пример визуализации алгоритма поиска в глубину. Рисунок представляет собой один из кадров, сгенерированный в процессе визуализации алгоритма. В левой части рисунка отображен текст алгоритма. Наличие атрибута цвета у вершины графа означает факт, что вершина уже была посещена в процессе обхода. Строка алгоритма может быть в одном из трех состояний: тонкий темный шрифт, светлый тонкий шрифт и светлый толстый шрифт. Первое состояние означает, что соответствующая строка алгоритма была исполнена хотя бы один раз. Второе состояние означает, что текущий визуальный эффект инициирован данной инструкцией. Последнее состояние означает, что инструкция еще не была исполнена. В правой части представлен иерархический помеченный граф с атрибутами. Наличие или отсутствие атрибута приводит к реализации соответствующих визуальных эффектов. В этом примере посещенные вершины получают установленный атрибут цвета. Такому значению атрибута соответствует увеличение толщины линии, применяемой для рисования окружности. Вершины с тонким контуром еще не были посещены.

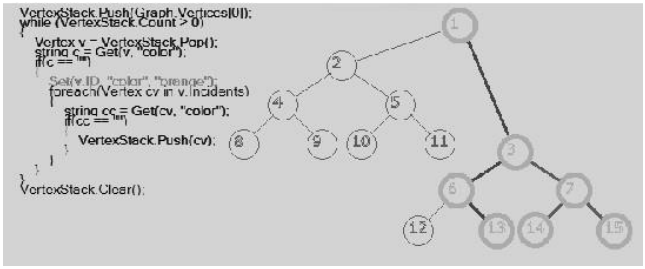

Рис. 3: Пример визуализации алгоритма поиска в глубину, один из промежуточных кадров.

При визуализации графовых алгоритмов существует возможность применить методы, повышающие наглядность визуализации. Если для некоторой записи истории определен контекст визуализации эффекта, то этот контекст используется для повышения наглядности работы алгоритма. Например, визуализации алгоритма обхода в глубину можно применять плавную визуализацию вдоль дуги соединяющей предыдущую и текущую вершины. В таком случае контекстом визуализации для текущей вершины будет предыдущая вершина. В случае, если предыдущая вершина не является смежной с текущей вершиной, для повышения наглядности визуализации применяется плавная визуализация вдоль дуг, принадлежащих кратчайшему пути, соединяющему текущую и предыдущую вершины. На рисунке 3 представлен пример использования визуализации с использованием контекста. В данном примере в процессе обхода, при продвижении от предыдущей вершины к текущей вершине изменяется толщина линий, применяемых для визуализации инцидентных дуг. Причем утолщенная светлая линия применяется для рисования дуг, принадлежащих пути от текущей вершины до корня дерева. Утолщенная темная линия применяется для рисования дуг, инцидентных уже посещенным вершинам.

Для повышения наглядности визуализации полезно отображать дополнительные структуры данных. например, при визуализации алгоритма обхода графа в ширину полезно отображать содержимое очереди. Содержимое очереди легко представляется в виде графа. Это, на ряду с использованием иерархических графов, позволяет отобразить содержимое очереди без дополнительных усилий. Так дерево иерархии графа будет содержать корень и два листа. Первый лист соответствует фрагменту входного графа, а второй лист соответствует фрагменту графа, изображающего очередь.

5. ЗАКЛЮЧЕНИЕ

В данной статье описывается модель визуализации алгоритмов на графах, обеспечивающая построение визуализаций алгоритмов за счет использования алгоритма в качестве параметра и системы визуальных эффектов. Также описывается метод для повышения наглядности визуализации алгоритмов на графах, позволяющий использовать дополнительную информацию, генерируемую во время исполнения алгоритма. Разработана система

визуализации алгоритмов, обеспечивающая апробацию на практике предлагаемого подхода визуализации алгоритмов на графах. Результатом работы системы является последовательность изображений, соответствующих промежуточным состояниям графовой модели во время работы алгоритма. Интерактивность визуализации достигается с помощью возможности изменять текст алгоритма и заново генерировать визуализацию, а также с помощью возможности гибко настроить интерпретацию имен атрибутов, используемыми в тексте, с помощью множества визуальных эффектов.

6. REFERENCES

[1] Касьянов В. Н., Евстигнеев В. А. *Графы в программировании: обработка, визуализация и применение. - СПб. БНВ-Петербург, 2003. - 1104 с.. ил. ISBN 5-94157-184-4.*

[2] Demetrescu C., Finocchi I., Stasko J. T. *Specifying Algorithm Visualizations: Interesting Events or State Mapping? // In Proc. of Dagstuhl Seminar on Software Visualization - Lect. Notes in Comput. Sci. - 2001. - P. 16-30.*

[3] Demetrescu C., Finocchi I. *A general-purpose logic-based visualization framework, Proceedings of the 7th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media (WSCG'99), pp. 55-62, Plzen, Czech Republic, February 1999.*

[4] Lisitsyn I.A., Kasyanov V.N. *Higres - visualization system for clustered graphs and graph algorithms // Proc. of Graph Drawing 99. - Berlin a.o.: Springer Verlag, 1999. - P. 82-89. - (Lect.Notes in Comput. Sci.; Vol. 1731).*

[5] Shaffer C. A., Cooper M. L., Alon A. J. D., Monika Akbar, Michael Stewart, Sean Ponce, S. H. Edwards. *Algorithm Visualization: The State of the Field // ACM Transactions on Computing Education TOCE. -- ACM, 2010. -- v. 10, is. 3, pp. 1-22.*

[6] Higres. *http://pcosrv.iis.nsk.su/higres/.*

[7] Leonardo. *http://www.dis.uniroma1.it/~demetres/Leonardo/.*