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

ФАКУЛЬТЕТ «Робототехники и комплексной автоматизации»
КАФЕДРА «Системы автоматизированного проектирования (РК-6)»

НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЕ ЗАМЕТКИ

по направлению «Разработка систем инженерного анализа и ресурсоемкого ПО (rndhpc)»

Авторы (исследователи):	Крехтунова Д., Ершов В., Муха В., Тришин И.
Научный(е) руководитель(и):	Соколов А.П.
Консультанты:	@Фамилия И.О.@

Москва, 2021–2021

RNDHPC
C59

Работа (документирование) над научным направлением начата 20 сентября 2021 г.

Руководители по направлению:

СОКОЛОВ, Александр Павлович	–	канд. физ.-мат. наук, доцент кафедры САПР, МГТУ им. Н.Э. Баумана
ПЕРШИН, Антон Юрьевич	–	PhD, ассистент кафедры САПР, МГТУ им. Н.Э. Баумана

Исследователи (студенты кафедры САПР, МГТУ им. Н.Э. Баумана):
Крехтунова Д., Ершов В., Муха В., Тришин И.

C59 **Крехтунова Д., Ершов В., Муха В., Тришин И.**. **Разработка систем инженерного анализа и ресурсоемкого ПО (rndhpc)**: Научно-исследовательские заметки. / Под редакцией Соколова А.П. [Электронный ресурс] — Москва: 2021. — 32 с. URL: https://arch.rk6.bmstu.ru (облачный сервис кафедры РК6)

Документ содержит краткие материалы, формируемые обучающимися и исследователями в процессе их работ по одному научному направлению.

Документ разработан для оценки результативности проведения научных исследований по направлению «Разработка систем инженерного анализа и ресурсоемкого ПО» в рамках реализации курсовых работ, курсовых проектов, выпускных квалификационных работ бакалавров и магистров, а также диссертационных исследований аспирантов кафедры «Системы автоматизированного проектирования» (РК6) МГТУ им. Н.Э. Баумана.

RNDHPC

Содержание

1 Теоретические основы графоориентированного программного каркаса

2022.01.01: Отличие сетевых моделей от сетевых графиков

> “*Сетевые модели отличаются от сетевых графиков тем, что в их вершинах могут реализовываться сложные логические и вероятностные функции, а также тем, что в них допускаются контуры*

Малинин Л.И.[1], 1970, [1].

В работе [1] проф. Л.И. Малинин использует термин *контуры*, что в современной литературе по теории графов часто называют *циклами*.

В работах [2, 3] д.т.н. В.И. Нечипоренко представляет обобщёный подход к графовому описанию сложных процессов и систем.

Подготовлено: *Соколов А.П. (РК-6), 2022.01.01*

2 Разработка графоориентированного дебаггера

2021.11.10: Автоматизированные и автоматические методы отладки, применяемые при реализации сложных вычислительных методов (СВМ), отладка наукоёмкого кода, science code debugging, graph based programming и пр. (первичный обзор литературы)

Анализ взвешенных графов вызовов для локализации ошибок программного обеспечения

Далее представлен материал, являющийся результатом анализа работы [4].

Проблема Обнаружение сбоев, которые приводят к ошибочным результатам с некоторыми, но не со всеми входными данными.

Предложенное решение Метод анализа графов вызовов функций с последующим составлением рейтинга методов, которые вероятнее всего содержат ошибку.

Описание решения *Граф вызовов*

Метод основан на анализе графа вызовов. Такой граф отражает структуру вызовов при выполнении конкретной программы. Без какой-либо дополнительной обработки граф вызовов представляет собой упорядоченное дерево с корнем.

[1] Профессор Малинин Л.И. использовал псевдоним и публиковался как Ермилов Л.И.

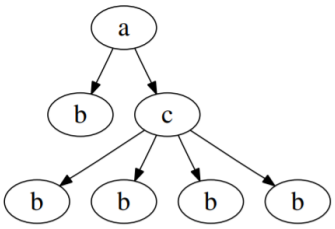


Рис. 1. Граф вызовов

Узлами являются сами методы, а гранями – их вызовы. Метод main() программы обычно является ее корнем, а все методы, вызываемые напрямую, являются ее дочерними элементами.

Редукция графа

Трассировка представляется в виде графов вызовов, затем повторяющиеся вызовы методов, вызванные итерациями, удаляются, вместо этого вводятся веса ребер, представляющие частоту вызовов.

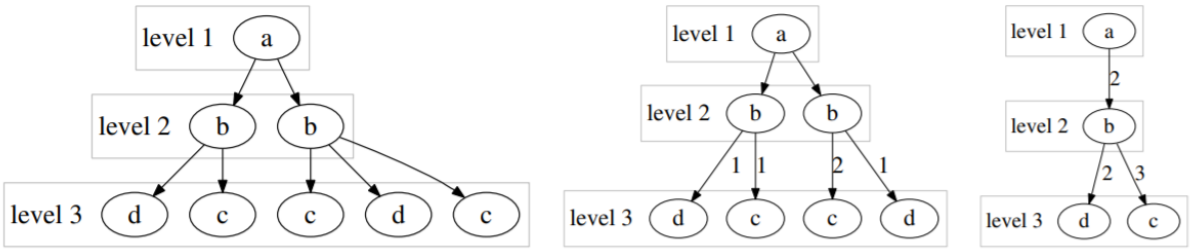


Рис. 2. Редукция графа вызовов

Поиск подграфов

Для поиска подграфов используется фреймворк для ранжирования потенциально ошибочных методов.

После сокращения графов вызовов, полученных от правильного и неудачного выполнения программ, применяется поиск часто встречающихся замкнутых подграфов SG в наборе данных графа G, используя алгоритм CloseGraph. Полученный набор подграфов разделяется на те, которые встречаются при правильном и неудачном выполнении (SGcf), и те, которые возникают только при неудачном выполнении (SGf).

Анализ

Два набора подграфов рассматриваются отдельно.

SGcf используется для построения рейтинга на основе различий в весах ребер при правильном и неудачном выполнении. Графы анализируются: применяется алгоритм выбора характеристик на основе энтропии к весам различных ребер для вычисления вероятности вызова метода и вероятности содержания в нем ошибки.

Алгоритм на основе энтропии не может обнаружить ошибки, которые не влияют на частоту вызовов и не учитывает подграфы, которые появляются только в ошибочной

версии (SGf).

Поэтому отдельно рассчитывается оценка для методов, содержащихся только в ошибочной версии. Эта оценка - еще одна вероятность наличия ошибки, основанная на частоте вызовов методов при неудачных выполнениях.

Затем вычисляется общая вероятность наличия ошибки для каждого метода. Этот рейтинг дается разработчику программного обеспечения, который выполняет анализ кода подозрительных методов.

Интегрированная отладка моделей Modelica

Далее представлен материал, являющийся результатом анализа работы [5].

Modelica – объектно-ориентированный, декларативный язык моделирования сложных систем (в частности, систем, содержащих механические, электрические, электронные, гидравлические, тепловые, энергетические компоненты).

Проблема Из за высокого уровня абстракции и оптимизации компиляторов, обеспечивающих простоту использования, ошибки программирования и моделирования часто трудно обнаружить.

Предложенное решение В статье представлена интегрированная среда отладки, сочетающая классическую отладку и специальные техники (для языков основанных на уравнениях) частично основанные на визуализации графов зависимостей.

Описание решения На этапе моделирования пользователь обнаруживает ошибку в нанесенных на график результатах, или код моделирования во время выполнения вызывает ошибку.

Отладчик строит интерактивный граф зависимостей (IDG) по отношению к переменной или выражению с неправильным значением.

Узлы в графе состоят из всех уравнений, функций, определений значений параметров и входных данных, которые использовались для вычисления неправильного значения переменной, начиная с известных значений состояний, параметров и времени.

Переменная с ошибочным значением (или которая вообще не может быть вычислена) отображается в корне графа.

Ребра могут быть следующих двух типов.

- Ребра зависимости данных: направленные ребра, помеченные переменными или параметрами, которые являются входными данными (используются для вычислений в этом уравнении) или выходными данными (вычисляются из этого уравнения) уравнения, отображаемого в узле.

- Исходные ребра: неориентированные ребра, которые связывают узел уравнения с реальной моделью, к которой принадлежит это уравнение. ребра, указывающие

из сгенерированного исполняемого кода моделирования на исходные уравнения или части уравнений, участвующие в этом коде

Пользователь может:

- Отобразить результаты симуляции, выбрав имя переменной или параметра(названия ребер). График переменной покажется в дополнительном окне. Пользователь может быстро увидеть, имеет ли переменная ошибочное значение.
- Отобразить код модели следуя по исходным ребрам.
- Вызвать подсистему отладки алгоритмического кода, если пользователь подозревает, что результат переменной, вычисленной в уравнении, содержащем вызов функции, неверен, но уравнение кажется правильным.

Используя эти средства интерактивного графа зависимостей, пользователь может проследить за ошибкой от ее проявления до ее источника.

Систематические методы отладки для масштабных вычислительных фреймворков высокопроизводительных вычислений

Далее представлен материал, являющийся результатом анализа работы [6].

Параллельные вычислительные фреймворки для высокопроизводительных вычислений играют центральную роль в развитии исследований, основанных на моделировании, в науке и технике.

Фреймворк Uintah был создан для решения сложных задач взаимодействия жидких структур с использованием параллельных вычислительных систем.

Проблема Поиск и исправление ошибок в параллельных фреймворках, возникающих из-за параллельного характера кода.

Предложенное решение В статье описывается подход к отладке крупномасштабных параллельных систем, основанный на различиях в исполнении между рабочими и нерабочими версиями. Подход основывается на трассировке стека вызовов.

Исследование проводилось для вычислительной платформы Uintah Computational Framework.

Описание решения Так как количество трассировок стека, которые можно получить при выполнении программы, может быть большим, для лучшего понимания используются графы, которые могут сжать несколько миллионов трассировок стека в одну управляемую фигуру. Метод основан на получении объединенных графов трассировки стека (CSTG).

Хотя сбор и анализ трассировки стека ранее изучались в контексте инструментов и подходов, их внимание не было сосредоточено на кросс-версии (дельта) отладке.

```
void A()                {
       cstg.addStackTrace();
}
void B()                {

     A();
}
int   main() {
    int     x = random();
    if       (x > 0)  B();
    A();
}
```

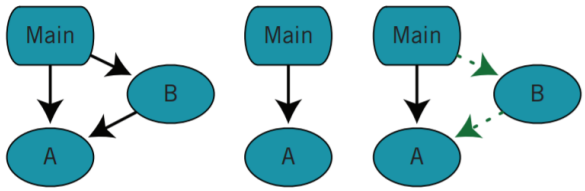


Рис. 3. Пример построения CSTG

CSTG не записывают каждую активацию функции, а только те, что в трассировках стека ведут к интересующей функции (функциям), выбранной пользователем. Каждый узел CSTG представляет все активации конкретного вызова функции. Помимо имен функций, узлы CSTG также помечаются уникальными идентификаторами вызова. Грани представляют собой вызовы между функциями.

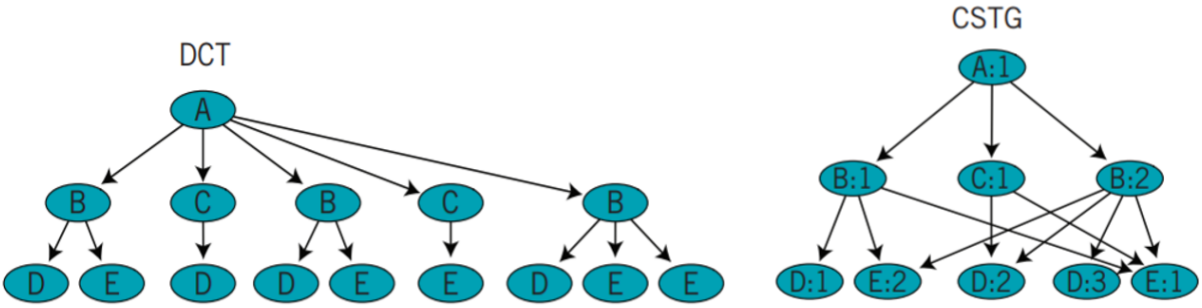


Рис. 4. CSTG

Пользователю необходимо вставить в интересующие функции вызовы cstg.addStackTrace(). Инструмент CSTG автоматически запускает тестируемый пример с использованием различных сценариев, создает графы и помогает пользователям увидеть существенные различия между сценариями. Сама ошибка обычно обнаруживается и подтверждается с помощью традиционного отладчика, при этом дельта-группы CSTG наводят на место возникновения ошибки.

Ориентированный на данные фреймворк для отладки параллельных приложений

Далее представлен материал, являющийся результатом анализа работы [7].

Проблема Обнаружение ошибок в крупномасштабных научных приложениях, работающих на сотнях тысяч вычислительных ядер.

Неэффективность параллельных отладчиков для отладки пета-масштабных приложений.

Предложенное решение В этом исследовании представлена реализация фреймворка для отладки, ориентированного на данные, в котором в качестве основы используются утверждения. Подход, ориентированный на данные можно использовать для повышения производительности параллельных отладчиков.

Метод, представленный в статье основан на параллельном отладчике Guard, который поддерживает тип утверждения во время отладки, называемые сравнительные утверждения.

Описание решения Утверждение – это утверждение о предполагаемом поведении компонента системы, которое должно быть проверено во время выполнения. В программировании программист определяет утверждение, чтобы гарантировать определенное состояние программы во время выполнения.

Пользователь делает заявления о содержимом структур данных, затем отладчик проверяет достоверность этих утверждений.

В следующем списке показаны примеры утверждений, которые можно использовать для обнаружения ошибок во время выполнения:

- «Содержимое этого массива всегда должно быть положительным».
- «Сумма содержимого этого массива всегда должна быть меньше постоянной границы».
- «Значение в этом скаляре всегда должно быть больше, чем значение в другом скаляре».
- «Содержимое этого массива всегда должно быть таким же, как содержимое другого массива».

В работе используются два новых шаблона утверждений помимо сравнительных: общие специальные утверждения и статистические утверждения.

Общие специальные утверждения позволяют использовать простые арифметические и булевы операции.

```
for (j = 0; j < n; j++) {
  for (i = 0; i < m; i++) {
    pold[j][i] = 50000.;
    pressure[j][i] = 50000.; }}
```

```
assert $a::pressure@"init.c":180 = 50000
```

Например, учитывая следующий фрагмент кода из исходного файла init.c и предполагая, что код вызывается с набором процессов $a, можно рассмотреть следующее утверждение:

Это утверждение гарантирует, что каждый элемент в структуре данных набора процессов $a равен 50 000 в строке 180 исходного файла init.c.

Соответственно, *сравнительные утверждения* позволяют сравнивать две отдельные структуры данных во время выполнения.

Следующее утверждение сравнивает данные из big_var в $a в строке 4300 исходного файла ref.c с large_var в $b в строке 4300 исходного файла sus.c.

```
assert $a::big_var@"ref.c":4300=$b::large_var@"sus.c":4300
```

Статистические утверждения - это определяемый пользователем предикаты, состоящие из двух моделей данных в форме либо статистических примитивов (средние значения, значения стандартного отклонения), либо функциональных моделей (гистограммы, функции плотности)

Статистические утверждения позволяют пользователю сравнивать информацию о шаблонах данных между двумя структурами данных, тогда как более ранние утверждения требовали сравнения точных значений.

Например, можно утверждать, что среднее значение большого набора данных находится между определенными границами до или после вызова функции или что количество элементов в массиве должно находиться в определенном диапазоне.

Определение степени и источников недетерминизма в приложениях MPI с помощью ядер графов

Далее представлен материал, являющийся результатом анализа работы [8].

Проблема Выявление причин сбоев воспроизводимости в приложениях на экзафлопсных платформах.

Крупномасштабные приложения MPI обычно принимают гибкие решения во время выполнения том порядке, в котором процессы обмениваются данными, чтобы улучшить свою производительность. Следовательно, недетерминированные коммуникативные модели стали особенностью этих научных приложений в системах высокопроизводительных вычислений.

Недетерминизм мешает разработчикам отслеживать вариации выполнения программы для отладки. Также сложно воспроизвести результаты при повторных запусках, что затрудняет доверие к научным результатам.

Предложенное решение Фреймворк для определения источников недетерминизма с использованием графов событий.

Описание решения Параллельное выполнение программы моделируется в виде ориентированных графов событий, ядра графов используются, чтобы охарактеризовать изменения межпроцессного взаимодействия от запуска к запуску. Ядра могут количественно определять тип и степень недетерминизма, присутствующего в моделях коммуникации MPI.

Фреймворк позволяет найти первопричины недетерминизма в исходном коде без знания коммуникативных паттернов приложения.

Платформа моделирует недетерминизм в следующие три этапа.

Этап первый. Сбор трассировки выполнения. Фреймворк фиксирует трассировку нескольких выполнений недетерминированного приложения с помощью двух модулей трассировки: CSMPI (фиксирует стек вызововов, связанных с вызовами функций MPI) и DUMPI (фиксирует порядок отправляемых и получаемых сообщений для каждого процесса MPI).

Для каждого выполнения приложения фреймворк генерирует один файл трассировки CSMPI и один файл трассировки DUMPI для каждого процесса MPI (или ранга). Файлы трассировки впоследствии загружаются конструктором графа событий для восстановления порядка сообщений выполнения.

Этап второй. Построение модели графа событий. На втором этапе фреймворк моделирует выполнение недетерминированного приложения в виде ориентированного ациклического графа (DAG), используя файлы трассировки, созданные на первом этапе.

Здесь вершины представляют собой связи point-to-point, такие как отправка и получение сообщения, а направленные ребра представляют отношения между этими событиями. Модели межпроцессного взаимодействия этой формы обычно называются графами событий.

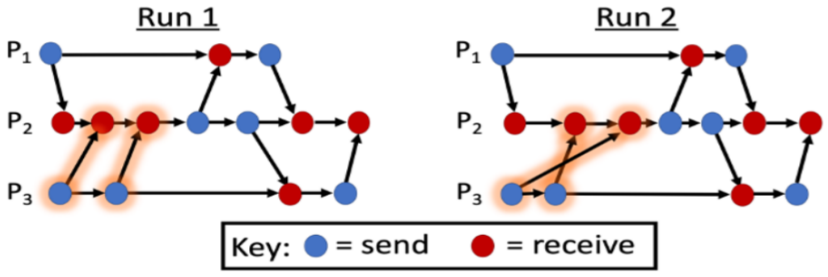


Рис. 5. DGA

Затем используются ядра графов для количественной оценки (несходства) графов событий, тем самым количественно оценивая степень проявления недетерминированности в приложении.

Ядра графов представляют собой семейство методов для измерения структурного сходства графов.

Простыми словами, ядро графа можно рассматривать как функцию, которая подсчитывает совпадающие подструктуры (например, поддеревья) двух входных графов, как показано на рис. 6, сопоставляя пары графов со скалярами, которые количественно определяют, насколько они похожи.

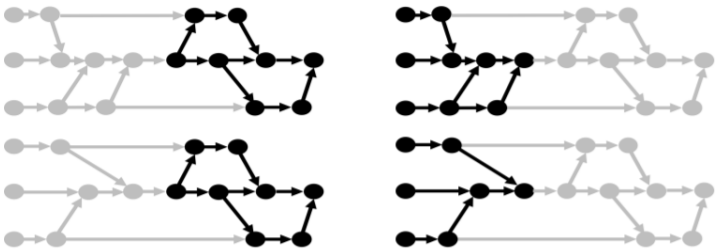


Рис. 6. Сравнение двух графов

Этап третий. Анализ графа событий. На последнем этапе анализ графа событий позволяет ученым количественно оценить недетерминированность многократного выполнения приложения MPI без каких-либо знаний о коммуникативных моделях приложения MPI.

Подготовлено: *Крехтунова Д.Д. (РК6-73Б), 2021.11.10*

2021.11.22: Web-инструменты для древовидного представления ассоциативных массивов, визуализация ассоциативных массивов (первичный обзор литературы)

Ассоциативные массивы

Ассоциативный массив – это массив, в котором обращение к значению осуществляется по ключу (того или иного типа).

Часто в качестве ключа используется не индекс, а строка, задаваемая программистом. Таким образом представить ассоциативный массив можно как набор пар “ключ-значение”. При этом каждое значение связано с определённым ключом.

Поддержка ассоциативных массивов есть во многих интерпретируемых языках программирования высокого уровня, таких, как Perl, PHP, Python, Ruby, Tcl, JavaScript и других.

Реализации ассоциативного массива Простейший способ хранения ассоциативных массивов – список пар (ключ, значение), где *ключ* определяет “индекс” элемента, а *значение* – значение элемента с этим индексом.

Наиболее популярными являются реализации ассоциативных массивов, основанные на различных деревьях поиска. Так, например, в стандартной библиотеке STL языка C++ контейнер *map*<*K*,*T*> реализован на основе красно-чёрного дерева [**?**]. В языках Java, Ruby, Tcl, Python используется один из вариантов хеш-таблицы [**?**]. Известны и другие реализации (ааа, ммм, ...).

Операции с деревом работают быстрее. При реализации на основе списков все функции требуют $O(n)$ операций, где n – количество элементов в рассматриваемой структуре. Операции над деревьями же требуют $O(h)$, где h – максимальная глубина дерева.

Данные в дереве хранятся в его вершинах. В программах вершины дерева обычно представляют структурой, хранящей данные и две ссылки на левого и правого сына.

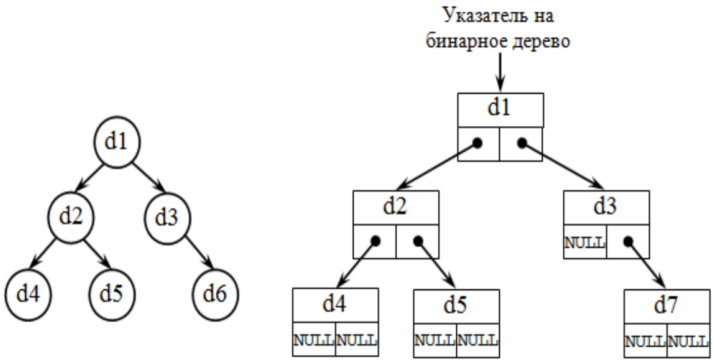


Рис. 7. Пример представления бинарного дерева

Набор библиотек TnT для web-визуализации деревьев и аннотаций на основе трассировок

В статье [9] представлен набор библиотек, предназначенных для web-визуализации деревьев и трассировок.

В статье говорится в первую очередь о визуализации биологических данных в веб-приложениях. Но представленные библиотеки не имеют узкой направленности и могут использоваться для разных целей.

Библиотеки TnT (англ. Trees and Tracks) предназначены для создания настраиваемых, динамических и интерактивных визуализаций деревьев и аннотаций на основе треков.

Библиотеки написаны на Javascript с использованием библиотеки D3, которая сама по себе является мощным инструментом визуализации данных. Она использует стандарты масштабируемой векторной графики (SVG), HTML и CSS.

Ниже приведено краткое описание библиотек TnT, непосредственно связанных с визуализацией деревьев.

- *TnT Tree*

Эта библиотека построена на основе кластерного расположения D3 (D3 cluster layout) и позволяет создавать динамические и интерактивные деревья. Он состоит из нескольких настраиваемых элементов: макета, определяющего общую форму дерева, узлов дерева в которых можно изменять форму, размер и цвет, ярлыков, состоящих из текста или изображений и данных для загрузки объектов Javascript или строк newick / nhx.2.2 TnT Tree Node.

PhyloCanvas - аналогичный проект, предлагающий средства для визуализации деревьев. В качестве основной технологии в нем используется Canvas. Но библиотеки TnT более универсальны и поддерживают интеграцию с другими библиотеками. Документацию и примеры для TnT Tree можно найти по ссылке http://tntvis.github.io/tnt.tree/.

- *TnT Tree Node*

  Эта библиотека предоставляет методы для управления деревом на уровне данных и используется TnT Tree, хотя также может использоваться и независимо. Методы, включенные в TnT Tree Node, варьируются от вычисления наименьшего общего предка набора узлов до извлечения поддеревьев. Документацию для этой библиотеки можно найти как часть документации библиотеки TnT Tree.

Программное обеспечение Empress интерактивного и исследовательского анализа многомерных наборов данных на основе деревьев

В работе [10] представлен интерактивный web-инструмент EMPress для визуализации деревьев в контексте микробиома, метаболома и других областей. EMPress предоставляет широкие функциональные возможности, такие как анимация объектов, наряду со стандартными функциями визуализации дерева.

EMPress реализован в виде подключаемого модуля QIIME 2 (или отдельной программы Python, которую можно использовать вне QIIME 2), способной создавать HTML-документы с автономным пользовательским интерфейсом визуализации. Кодовая база состоит из компонента Python и компонента JavaScript. Кодовая база Python отвечает за проверку данных, предварительную обработку, фильтрацию и форматирование. Взаимодействие с пользователем, рендеринг и генерация рисунков обрабатываются базой кода JavaScript.

Кодовая база Python EMPress использует такие модули, как NumPy, SciPy, Pandas, Click, Jinja2, scikit-bio, формат BIOM и EMPeror.

В кодовой базе JavaScript используются Chroma.js, FileSaver.js, glMatrix, jQuery, Require.js, Spectrum, и Underscore.js.

Программный инструмент Treemap визуализации иерархических структур

В статье [11] представлена интерактивная версия Treemap.

В теории графов дерево – это особый тип графа, который связан и ацикличен [**?**]. Обычно деревья представляются визуально с помощью диаграмм узлов и связей

(древовидных диаграмм). В деревьях ребра присутствуют только между соседними слоями, и, следовательно, деревья наилучшим образом подходят для представления иерархических данных, для которых характерно расположение элементов на различных уровнях относительно друг друга: “ниже”, “выше” или “на одном уровне”.

Treemap - это метод визуализации иерархических структур. Используя этот метод, можно отобразить дерево с миллионами узлов в ограниченном пространстве. Основная идея, лежащая в основе этого метода визуализации, состоит в том, чтобы выделиять прямоугольники для родительских узлов, а для дочерних узлов блоки (прямоугольники) в соответствующем родительском прямоугольнике.

Интерфейс создавался с использованием html / css с jQuery для обработки событий, связанных с кликами.Входными данными для интерфейса является дерево родительских указателей (parent pointer tree - структура данных N-арного дерева, в которой каждый узел имеет указатель на свой родительский узел, но не указывает на дочерние узлы).

Web-инструмент DoubleRecViz для визуализации согласования деревьев транскриптов и генов

В статье [12] представлен веб-инструмент для визуализации согласований между филогенетическими деревьями (деревья, отражающее эволюционные взаимосвязи между различными видами, имеющими общего предка).

В статье описан формат DoubleRecViz, основанный на формате phyloXML (XML, предназначенный для описания филогенетических деревьев и связанных с ними данных).

DoubleRecViz написан на Python и использует библиотеку Dash, которая предоставляет функции динамической визуализации веб-данных. Dash является связкой Flask, React.Js, HTML и CSS.

Приложения на Dash — веб-серверы, которые запускают Flask и связывают пакеты JSON через HTTP-запросы. Интерфейс Dash формирует компоненты, используя React.js.

Компоненты Dash — это классы Python, которые кодируют свойства и значения конкретного компонента React и упорядочиваются как JSON.

Полный набор HTML-тегов также обрабатывается с помощью React, а их классы Python доступны через библиотеку. CSS и стили по умолчанию хранятся вне базовой библиотеки, чтобы сохранить принцип модульности и независимого управления версиями.

Подготовлено: *Крехтунова Д.Д. (РК6-73Б), 2021.11.22*

2021.12.19: Концептуальная постановка задачи

Требуется реализовать web-ориентированный программный инструмент (далее *GBSE-отладчик*), обеспечивающий проведение отладки графовых реализации некоторых сложных вычислительных методов. GBSE-отладчик должен обеспечивать отслеживание текущих значений каждого отдельного элемента данных в состояниях, соответствующих узлам связанной графовой модели.

Цель разработки. Создать программный инструментарий (web-ориентированный), который бы позволил визуализировать значения отдельных элементов общих данных[13], остановив обработку на произвольном, выбираемом(ых) заранее, состоянии(ях) данных.

Назначение. Реализация задачи разработки GBSE-отладчика позволит отслеживать текущие значения отдельных элементов данных в произвольном состоянии данных (узле) при проведении текущего расчета, что упростит процесс разработки графоориентированных решателей.

Поставленные задачи (частичный перечень).

1. Провести обзор литературы по темам:
   (a) "Автоматические методы отладки наукоемкого кода" (автоматизированные и автоматические методы отладки, приме- няемые при реализации сложных вычислительных методов (СВМ), отладка наукоёмкого кода, science code debugging, graph based programming);
   (b) "Методы визуализации ассоциативных массивов" (web-инструменты для древовидного представления ассоциативных массивов, визуализация ассоциативных массивов).
2. Определить перечень поддерживаемых типов элементов данных СВМ, предложить и реализовать методы визуализации для каждого из них.
3. Разработать функцию на языке Python, позволяющую определять тип данных, хранящихся в каждом отдельном элементе ассоциативного массива (хранение данных осуществляется в объекте типа dict).
4. С использованием программного каркаса Django разработать программное обеспечение для визуализации объектов типа *dict* в древовидном виде.

Подготовлено: *Крехтунова Д.Д. (РК6-73Б), Соколов А.П. (РК6), 2021.12.19*

3 Разработка web-ориентированного редактора графовых моделей

2021.10.05: Обзор языка описания графов DOT

Язык описания графов DOT предоставляется пакетом утилит Graphviz (Graph Visualization Software). Пакет состоит из набора утилит командной строки и программ с графическим интерфейсом, способных обрабатывать файлы на языке DOT, а также из виджетов и библиотек, облегчающих создание графов и программ для построения графов. Более подробно будет рассмотрена утилита dot.

Замечание 1. dot – программный инструмент для создания многоуровневого графа с возможностью вывода изображения полученного графа в различных форматах (PNG, PDF, PostScript, SVG и др.).

Установка graphviz:

Linux: sudo apt install graphviz

MacOS: brew install graphviz

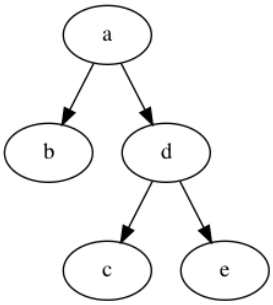
Вызов всех программ Graphviz осуществляется через командную строку, в процессе ознакомления с языком использовалась следующая команда:

dot −Tpng <pathToDotFile> −o <imageName>

В результате выполнения этой команды будет создано изображение графа в формате png.

Пример описания простого графа на языке DOT представлен далее.

```
digraph G {
    a -> b;
    a -> d -> c;
    d -> e;
}
```



Более подробная информация с примерами представлена в обзоре литературы, который размещён по следующему адресу:

01 - Курсовые проекты/2021-2022 - Разработка web-ориентированного редактора графовых моделей /0 - Обзор литературы/

Подготовлено: *Ершов В. (РК6-72Б), 2021.10.05*

2021.12.04: Краткое описание алгоритма визуализации графа

В ходе разработки web-ориентированного редактора графов, который позволяет импортировать и экспортировать файлы в формате aDOT [14], был разработан алгоритм визуализации графов. В этой заметке будет рассмотрен этот алгоритм и приведено

несколько примеров aDOT файлов, которые корректно визуализируются, используя этот алгоритм.

Теоретические основы и назначение графоориентированной программной инженерии представлены в работе [13]. Принципы применения графоориентированного подхода зафиксированы в патенте [15].

Один из самых важных критериев построенного графа – это его читаемость. Граф должен быть построен корректно и недопускать неоднозначностей при его чтении. Например, вершины не должны налегать друг на друга, ребра не должны пересекаться, создавая сложные для восприятия связи.

Проведя длительную аналитическую работу, было принято решение разбивать граф по уровням. Рассмотрим следующее aDOT-определение графовой модели G (листинг 1).

Листинг 1. Пример aDOT-определение простейшей графовой модели G

```
digraph G {
// Parallelism
  s1 [parallelism=threading]
// Graph definition
  __BEGIN__ ->s1
  s4 ->s6 [morphism=edge_1]
  s5 ->s6 [morphism=edge_1]
  s1 =>s2 [morphism=edge_1]
  s1 =>s3 [morphism=edge_1]
  s2 ->s4 [morphism=edge_1]
  s3 ->s5 [morphism=edge_1]
  s6 ->__END__
}
```

Если нарисовать такой граф на бумаге, то становится очевидно, что каждый набор вершин имеет одну координату по оси X, а связанные с ними вершины находятся правее по координате X, таким образом напрашивается разбиение графа по уровням. На рисунке (8) представлен этот граф, разбитый на уровни.

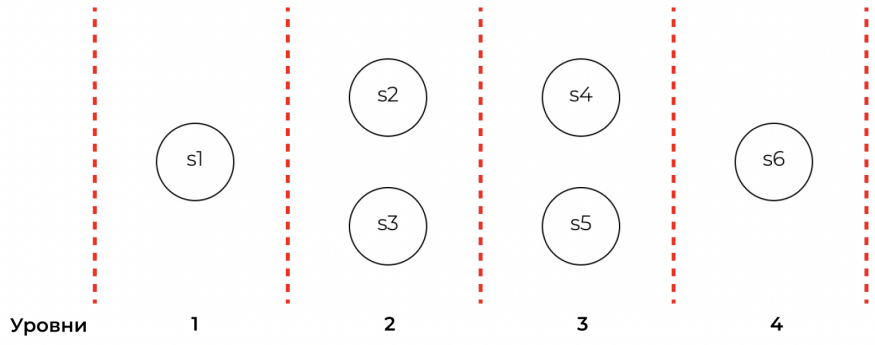


Рис. 8. Узлы графовой модели G, представленные на разных “уровнях”

Разбиение графа по уровням является основой алгоритма визуализации, далее на примере более сложного графа поэтапно разберем алгоритм.

Рассмотрим следующий файл на языке aDOT (листинг 2).

Листинг 2. Пример aDOT-определения графовой модели TEST

```
digraph TEST
{
// Parallelism
  s11 [parallelism=threading]
  s4 [parallelism=threading]
  s12 [parallelism=threading]
  s15 [parallelism=threading]
  s2 [parallelism=threading]
  s8 [parallelism=threading]
// Functions
  f1 [module=DEFAULT_VALUE, entry_func=DEFAULT_VALUE]
// Predicates
  p1 [module=DEFAULT_VALUE, entry_func=DEFAULT_VALUE]
// Edges
  edge_1 [predicate=p1, function=f1]
// Graph model description
  __BEGIN__ ->s1
  s6 ->s8 [morphism=edge_1]
  s7 ->s8 [morphism=edge_1]
  s10 ->s8 [morphism=edge_1]
  s11 =>s8 [morphism=edge_1]
  s11 =>s9 [morphism=edge_1]
  s14 ->s9 [morphism=edge_1]
  s3 ->s9 [morphism=edge_1]
  s4 =>s6 [morphism=edge_1]
  s4 =>s7 [morphism=edge_1]
  s4 =>s10 [morphism=edge_1]
  s4 =>s11 [morphism=edge_1]
  s12 =>s4 [morphism=edge_1]
  s12 =>s14 [morphism=edge_1]
  s13 ->s3 [morphism=edge_1]
  s15 =>s14 [morphism=edge_1]
  s15 =>s14 [morphism=edge_1]
  s2 =>s12 [morphism=edge_1]
  s2 =>s13 [morphism=edge_1]
  s2 =>s15 [morphism=edge_1]
  s1 ->s2 [morphism=edge_1]
  s8 =>s9 [morphism=edge_1]
  s8 =>s6 [morphism=edge_1]
  s9 ->__END__
}
```

В результате визуализации модели будет получен граф, представленный на рисунке (9).

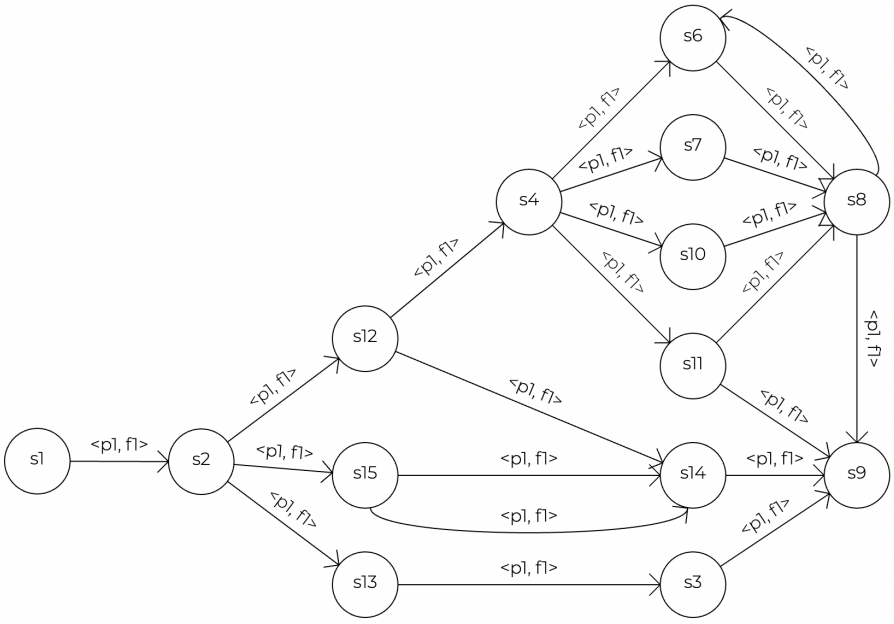


Рис. 9. Визуализация графовой модели TEST

Основная задача алгоритма – это отрисовать вершины графа в корректных позициях, а затем построить между ними ребра. Поскольку редактор графов – это web-ориентированное приложение, то вся бизнес-логика написана на языке JavaScript. В JavaScript имеется очень удобный инструмент – объекты. Аналогами объектов в других языках программирования являются хэш-карты, но в Javascript возможности использования объектов гораздо шире. Всю информацию об уровнях графа будем хранить в объекте levels. Ниже представлено как выглядит объект levels для рассматриваемого выше графа:

```
{
"1":[{"s1":[]}],
"2":[{"s2":["s1"]}],
"3":[{"s12":["s2"]}, {"s13":["s2"]}, {"s15":["s2"]}],
"4":[{"s4":["s12"]}],
"5":[{"s9":["s14","s3"]}, {"s6":["s4"]}, {"s7":["s4"]}, {"s10":["s4"]},
{"s11":["s4"]},{"s14":["s12","s15"]}, {"s3":["s13"]}],
"6":[{"s8":["s6","s7","s10","s11"]}, {"s9":["s11","s14","s3"]}]
}
```

Ключами объекта levels являются номера уровней, представленные в формате string. Свойсвами являются массивы, на один уровень - один массив. Каждый элемент масси-

ва содержит информацию об одной вершине на этом уровне, следовательно количество вершин на уровне - это размер массива. Далее заметим, что элемент массива это не просто string с названием вершины, а объект. Этот объект содержит один ключ - название вершины на этом уровне, а свойством является массив, который содержит список вершин из которых перешли в эту вершину (переход только по одному ребру).

В качестве примера рассмотрим уровень 5 объекта levels, который был представлен ранее.

```
{
  "5":[{"s9":["s14","s3"]}, {"s6":["s4"]}, {"s7":["s4"]},
  {"s10":["s4"]}, {"s11":["s4"]}, {"s14":["s12","s15"]}, {"s3":["s13"]}],
}
```

Уровень 5 в графе представлен 7-ю вершин: “s9”, “s6”, “s7”, “s10”, “s11”, “s14”, “s3”. Например, в вершину “s6” мы пришли из вершины “s4”, таким образом, по ключу “s6” находится массив с один элементом “s4”

```
{"s6": ["s4"]}
```

На рисунке (10) представлен граф с выделенным уровнем №5.

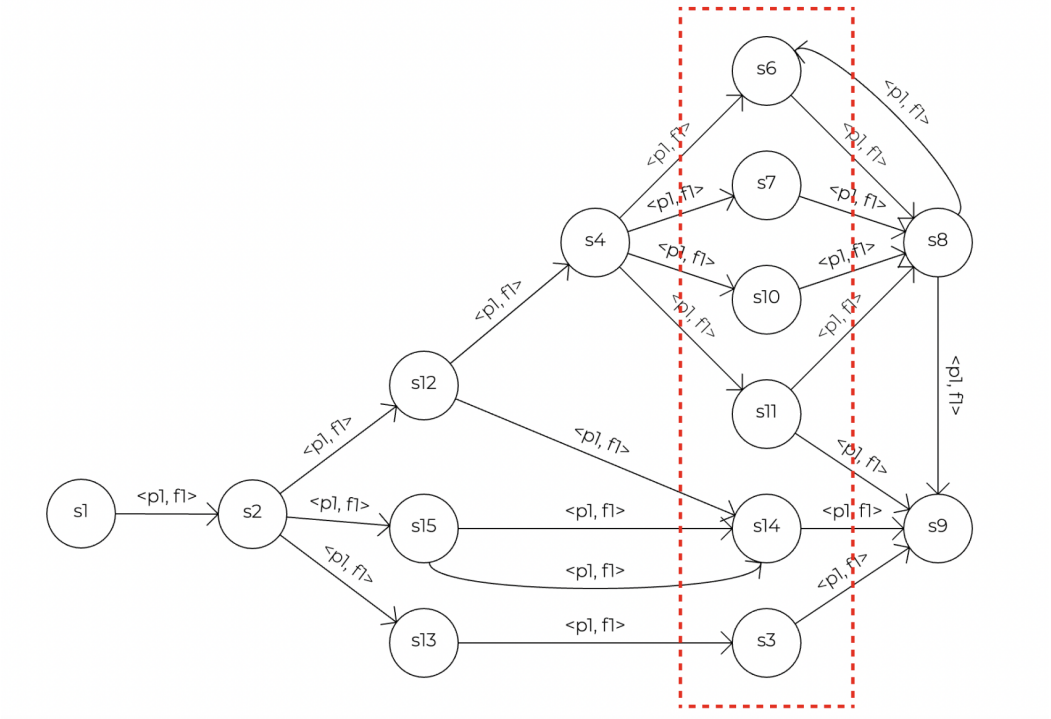


Рис. 10. Уровень №5 на графе

Обратим вниманием на то, что в объекте levels для уровня №5 содержится вершина s_9 которой нет на уровне №5, а она присутствует только на уровне №6. Заполнение объекта levels происходит слево-направо, то есть от меньшего уровня к большему, например, в графе представленном выше есть связь $s_{13} \rightarrow s_3$, таким образом пока в

объект levels не будет записана вершина s_{13}, мы не сможем записать связанную с ней вершину s_3.

Обратным образом происходит дублирование вершин, если обратиться к описанию графовой модели предсталвенной выше, то можно заметить такую связь: $s_1 \rightarrow s_2 \rightarrow s_{13} \rightarrow s_3 \rightarrow s_9$. При начальной инициализации объекта levels s_1 будет находиться на уровне 1, s_2 будет находиться на уровне 2 и так далее. Таким образом, вершина s_3 будет находиться на уровне 4, что не совсем корректно. Обработка таких ситуаций является углублением в реализацию алгоритма и в этой заметке не будет подробно описываться.

Для разрешения подобных коллизий после начальной инициализации объекта levels “в лоб” предусмотрено множество дополнительных проверок. Таким образом, на выходе мы получаем корректно сформированный объект levels и дополнительно формируем объект, который будет хранить информацию о вершинах которые находятся не на своем уровне, эти вершины не будут отрисовываться, но при этом они будут учитываться при выравнивании связанных с ними вершин, следовательно просто удалить такие вершины из объекта levels нельзя.

Сама визуализация вершин после формирования объекта levels также является темой отдельной заметки. На данном этапе работы над проектом сначала строится уровень содержащий наибольшее количество вершин, затем строятся оставшиеся уровни: сначала влево до уровня №1, затем вправо до самого последнего уровня.

Подготовлено: *Ершов В. (РК6-72Б), 2021.12.04*

2021.12.05: Описание текущего состояния проекта

В данной заметке описаны все основные пользовательские сценарии web-ориентированного редактора графов по состоянию на 2021.12.05. Для каждого сценария составлен следующий список:

1) реализованные возможности сценария;

2) сложности, возникшие при реализации сценария;

3) будущий функционал, который расширит user expirience.

Основые сценарии приложения:

1) создание графа в приложении путем добавления вершин и ребер между ними;

2) экспортирование графа в файл на языке описания графов aDOT;

3) импортирование графа из файла на языке описания графов aDOT.

Рис. 11. Создание вершины

Добавление вершины Для доблавения вершины необходимо выбрать соответствующий пункт в меню, а затем кликнуть на холст. После этого потребуется уточнить название вершины, после ввода названия вершины построение будет полностью завершено. Процесс построения вершины представлен на рисунке 11.

Дополнительный функционал сценария:

1) блокировка создания вершины если она находится в близости к другой вершине;
2) блокировка создания названия вершины, если такое название присвоено существующей вершине.

Сложности при реализации сценария:

1) валидировать все необходимые параметры в процессе создания вершины: положение вершины, название вершины;
2) выбор информации о вершине, которую нужно сохранить в оперативной памяти для дальнейшего использования в других сценариях.

Будущее расширение функционала сценария.

1) Возможность изменить положение вершины в процессе ее создания. На данный момент вершина создается в месте клика на холст (в том случае если местоположение вершины валидно) и это местоположение уже никак нельзя изменить.

2) Использование набора горячих клавиш для ускорения процесса создания вершины.

3) При названия подсказать пользователю название вершины, в том случае если прошлые названия имеют инкрементирующийся для каждой вершины постфикс, например, уже созданы вершины s_1, s_2, при создании новой вершины в поле ввода названия система предложит пользователю название s_3.

Добавление ребра Для добавления ребра необходимо поочередно выбрать две вершины, которые будут соединены ребром. Затем система потребует ввода предиката и функции, в том случае если введенный предикат или функция являются уникальными в рамках графа, то система потребует ввода информации о module и entry func. После этого построение ребра будет завершено. Процесс построения ребра представлен на рисунке 12.

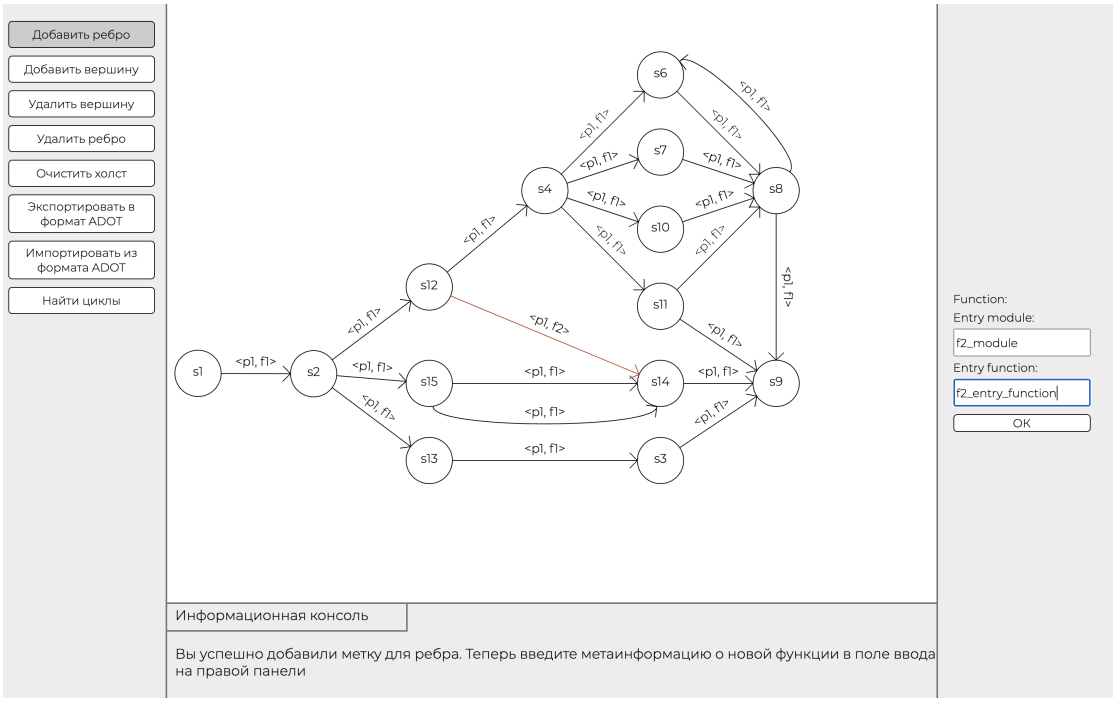


Рис. 12. Создание ребра

Дополнительный функционал сценария.

- Блокировка создания ребра из вершины в нее же саму.
- Если между вершинами уже существуют ребра, то система построит ребро, используя кривые Безье.
- Если на момент построения ребра из вершины выходит только одно ребро, то система потребует учтонения типа параллелизма (формат aDOT).

- Если предикат или функция не являются уникальными в рамках графа, то не требуется уточнение информации о module и entry func.

Сложности при реализации сценария.

- Использование множества вспомогательных функций: подсчет количества ребер между вершинами, для определения типа построения ребра (прямое или кривые Безье), подсчет количества ребер выходящих из вершины для уточнения типа параллелизма, проверка полей ввода предиката и функции (предикат может быть неопределен, а функция должна быть определена), проверка введенных предиката и функции на уникальность в рамках графа.

- Выбор типа построения ребра: прямое, кривые Безье. Обработка ситуации, когда пользователь создает цикл между этими вершинами.

Будущее расширение функционала сценария.

- Возможность перевыбора вершин для построения ребра между ними. На данный момент нельзя допустить ошибку при выборе вершин, необходимо закончить процесс построения ребра, удалить его и создать новое, выбрав другие вершины.

- Алгоритм для построения ребра с использованием кривых Безье. На данный момент ребро строится с помощью одной кривой Безье, что не позволяет строить ребра более сложного вида, тем самым разрешая коллизии, когда ребро проходит через другие вершины.

Удаление вершины Для удаления вершины необходимо выбрать соответствующий пункт в меню, а затем кликнуть на вершину для удаления - вершина удаляется с холста вместе со всеми связанными с ней ребрами.

Сложности при реализации сценария.

- Корректно удалить все связанные с вершиной ребра, а затем удалить информацию об этих ребрах и связанных вершин.

Будущее расширение функционала сценария.

- Возможность одновременного выбора нескольких вершин для удаления.

- Возможность откатить действия в том случае если была удалена лишняя вершина. В целом это касается всего приложения, на данный момент любое действие неотвратимо.

Удаление ребра Для удаления ребра необходимо выбрать соответствующий пункт в меню, а затем клкинуть на ребро для удаления - ребро удаляется с холста.

Сложности при реализации сценария.

- Удалить информацию о ребре из связанных этим ребром вершин.

Будущее расширение функционала сценария.

- Более корректный выбор ребра, на данный момент надо кликать ровно на ребро поскольку для перехвата события клика используется event.target.closest.
- Возможность одновременного выбора нескольких ребер для удаления.

Экспорт в формат aDOT Для экспорта в формат aDOT необходимо выбрать соответствующий пункт в меню, а затем поочередно выбрать две вершины, которые будут являться стартовой и конечной вершиной соответственно. Затем сформируется текстовый файл с описание графа в формате aDOT и автоматически начнется его загрузка.

Импорт из формата aDOT Для импорта из формата aDOT необходимо выбрать соответствующий пункт в меню, а затем выбрать файл для импорта. Далее система сама построит граф, сохранив всю необходимую информацию.

Сложности при реализации сценария.

- Разработка алгоритма визуализации...

Будущее расширение функционала сценария.

- Разработка более гибкого алгоритма визуализации, который сможет работать с более сложными графами.

Подготовлено: *Ершов В. (РК6-72Б), 2021.12.05*

4 Реализация поддержки различных стратегий распараллеливания в рамках графоориентированного программного каркаса

2021.11.14: Известные алгоритмы поиска циклов в ориентированных графах

Существует несколько групп алгоритмов поиска[16]:

1) алгоритмы, основанные на обходе ориентированного графа (ОГ);
2) алгоритмы, основанные на использовании матриц смежности, описывающих конкретный граф.

Одним из представителей алгоритмов первой группы (обхода ОГ) является алгоритм поиска в глубину (англ., depth-first-search (DFS)), который предполагает, что обход осуществляется из фиксированного и заранее заданного узла. Если число узлов ОГ равно n, то сложность алгоритма DFS $O(n^2)$ и $O(n^3)$ – для случая "многократного обхода" из каждого узла[2]. Поэтому для ускорения алгоритма необходимо применение модификаций, например распараллеливание алгоритма поиска в глубину [17].

Зададим граф $G(V, E)$, где $V = \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \ldots\}$ – множество вершин, $|V| = n$, а $E = \{\alpha\beta : \alpha\beta \in V \times V\}$ – множество рёбер (пример, $\{\boldsymbol{ab}, \boldsymbol{ad}, \ldots, \boldsymbol{ge}\}$).

Для определения ОГ, часто, применяют понятие матриц смежности[18]:

$$\|A\| = \|a_{ij}\|_{n\times n};$$
$$a_{ij} = \begin{cases} 1, & \text{– определено ребро с началом в узле } i \text{ и концом в узле } j; \\ 0, & \text{– ребро не определено.} \end{cases} \tag{1}$$

$$A = \begin{bmatrix} & \boldsymbol{a} & \boldsymbol{b} & \boldsymbol{c} & \boldsymbol{d} & \boldsymbol{e} & \boldsymbol{f} & \boldsymbol{g} \\ \boldsymbol{a} & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ \boldsymbol{b} & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ \boldsymbol{c} & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \boldsymbol{d} & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ \boldsymbol{e} & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \boldsymbol{f} & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \boldsymbol{g} & 0 & 0 & 1 & 1 & 1 & 01 & 0 \end{bmatrix} \tag{2}$$

Пример графа, построенного на основе его матрицы смежности A, приведен на рис. 13.

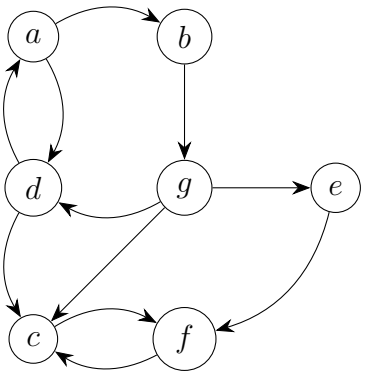


Рис. 13. Граф, построенный по матрице смежности A

Элемент матрицы смежности a_{ij} указывает на факт наличия определённого рёбра с началом в узле i и концом в узле j, тогда как транспонированная матрица A^T задаёт новый ОГ с теми же узлами и рёбрами, противоположно направленными относительно исходного ОГ.

[2]Для случая ОГ возможность идентификации всех циклов требует осуществления многократных обходов, начиная с каждого из узлов ОГ.

$$A^T = \begin{bmatrix} & \boldsymbol{a} & \boldsymbol{b} & \boldsymbol{c} & \boldsymbol{d} & \boldsymbol{e} & \boldsymbol{f} & \boldsymbol{g} \\ \boldsymbol{a} & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \boldsymbol{b} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \boldsymbol{c} & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ \boldsymbol{d} & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ \boldsymbol{e} & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ \boldsymbol{f} & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ \boldsymbol{g} & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3}$$

Введём обозначение матрицы, возведённой в степень m:

$$A^m = \underbrace{A \times A \times \ldots \times A}_{m}. \tag{4}$$

Матрица A^m обладает следующим свойством: элемент матрицы a^i_j равен числу путей из вершины i в вершину j, состоящих ровно из m рёбер[18]. Длиной цикла называют количстево рёбер в этом цикле. Для того, чтобы определить циклы длины $2m$, нужно найти матрицу циклов C_m, определяемую следующим образом:

$$C_m = A^m \wedge (A^T)^m; \tag{5}$$

где $\wedge$ – оператор конъюнкции, применяемый поэлементно.

Таким образом, матрица C_m будет содержать все циклы длины $2m$. Пример для поиска циклов длины 2. 6.

$$C_2 = A \wedge A^T = \begin{bmatrix} & \boldsymbol{a} & \boldsymbol{b} & \boldsymbol{c} & \boldsymbol{d} & \boldsymbol{e} & \boldsymbol{f} & \boldsymbol{g} \\ \boldsymbol{a} & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \boldsymbol{b} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \boldsymbol{c} & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \boldsymbol{d} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \boldsymbol{e} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \boldsymbol{f} & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \boldsymbol{g} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6}$$

Подготовлено: *Муха В. (РК6-73Б), 2021.11.14*

5 Графоориентированная методология разработки средств взаимодействия пользователя в системах автоматизированного проектирования и инженерного анализа

2021.11.06: Особенности применения графового описания процессов обработки данных в pSeven (DATADVANCE)

pSeven – это платформа для анализа данных, оптимизации и создания аппроксимационных моделей, дополняющая средства проектирования и инженерного анализа. pSeven позволяет интегрировать в единой программной среде различные инженерные приложения, алгоритмы многодисциплинарной оптимизации и инструменты анализа данных для упрощения принятия конструкторских решений[19].

Принципы функционирования

На концептуальном уровне в pSeven вводятся следующие понятия:

- Проект – набор файлов, используемых в pSeven для описания решений одной или нескольких задач и хранения результатов их решения.
- Расчётная схема (workflow) – формальное описание процесса решения некоторой задачи в виде ориентированного графа, узлами которого являются блоки, а рёбрами - связи. Такое описание хранится в бинарном файле с расширением .p7wf, использующем некоторый специализированный формат хранения подобного рода описаний.
- Блок – функциональный элемент расчётной схемы, отвечающий за обработку входных данных и формирование выходных данных.[20]
- Порт – переменная определённого типа, описанная в блоке и имеющая в нём уникальное имя, значение которой может быть передано в другие блоки или получено от них через связи.
- Связь (link) – одностороннее соединение между двумя портами, обеспечивающее передачу данных от одного к другому.

Проекты в pSeven имеют единую базу данных, куда записываются все результаты запусков расчётных схем и откуда берутся данные для их последующей презентации пользователю и их анализа. Для определения переменных, значения которых должны быть записаны в неё записаны, предусмотрены специализированные порты для самих расчётных схем, с которыми связываются те блоки, результаты выполнения которых интересуют пользователя.

Связи служат для маршрутизации данных. С их помощью осуществляется и взаимодействие между блоками и, кроме того, определяется очерёдность их запуска. В момент добавления связи в пакете pSeven выполняется проверка портов на совместимость. Они считаются совместимыми, если тип данных источника можно преобразовать к типу данных адресата.[20]

Поддержка циклов и ветвлений

Расчетная схема может включать расчетные циклы. Для их создания применяются специализированные блоки, имеющие функциональную возможность управления запуском других блоков в теле цикла и принятия решения о его прекращении. Такие блоки называются управляющими блоками циклов. Одним из характерных примеров является оптимизационный цикл, управление которым осуществляется из блока Optimizer [20], позволяющего, например, настроить максимальное число итераций или требуемую точность, как условия окончания.

Если речь идёт о задачах анализа данных, существует отдельный блок, обозначающий входную точку цикла (Loop). Такой цикл принимает на вход список наборов аргументов, каждый из которых будет обработан на соответствующем шаге цикла, и выдаёт список наборов выходных значений, которые сохраняются в базе данных проекта.

Кроме того, существует возможность включения в расчётную схему условного и безусловного ветвления. Первое достигается засчёт создания связей для подключения одного и того же порта вывода к различным портам ввода. В этом случае по каждой связи передается копия выходных данных, полученных у источника [20]. Условные ветвления создаются с помощью специального блока Condition, который по определённому условию передаёт входные данные одному из подключенных блоков. Их целесообразно использовать для устранения ошибок в работе блока, отбраковки некорректных входных данных и других аналогичных целей[20].

Особенности выполнения расчётных схем

При выполнении расчётных схем каждый блок запускается в отдельном процессе на уровне операционной системы. Как сказано ранее, начало выполнения блока определяется его связями с другими. Любой блок будет ожидать завершения работы другого блока только в том случае, если ему необходимо получить от него входные данные. Это означает, что два блока, не имеющих связей друг с другом Блоки, входящие в состав различных ветвлений расчетной схемы, могут запускаться параллельно, поскольку они не зависят друг от друга по используемым данным[20].

Кроме того, при обработке больших выборок данных может потребоваться обрабатывать по несколько наборов данных одновременно в независимых потоках исполнения. Помимо прочего этой цели служит блок Composite, который является контейнером для нескольких блоков. В его настройках можно включить опцию параллельного исполнения, указать, с какого порта данные будут обрабатываться параллельно, и указать максимальное число потоков. При запуске расчетной схемы пакет pSeven создает

несколько виртуальных экземпляров блока Composite и автоматически распределяет входные наборы данных между ними. Как только в одном из таких виртуальных блоков завершается расчет, он получает из выборки следующий набор для обработки.[21]

Подготовлено: *Тришин И.В. (РК6), 2021.11.06*

Список литературы

[1] Ермилов Л.И. Синтез сетевых моделей сложных процессов и систем. Москва: МО СССР, 1970. С. 100.

[2] Нечипоренко В.И. Структурный анализ и методы построения надёжных систем. Москва: Издательство «Советское радио», 1968. С. 256.

[3] Нечипоренко В.И. Структурный анализ систем (эффективность и надёжность). Москва: Издательство «Советское радио», 1977. С. 216.

[4] Frank Eichinger, Klemens Bohm, and Matthias Huber. Mining Edge-Weighted Call Graphs to Localise Software Bugs // Lecture Notes in Computer Science. 2008. P. 333–348.

[5] A. Pop, M. Sjolund, A. Asghar, P. Fritzson, F. Casella. Integrated Debugging of Modelica Models // Modeling, Identification and Control. 2014. Vol. 35, no. 2. P. 93–107.

[6] Alan Humphrey, Qingyu Meng, Martin Berzins, Diego Caminha B. de Oliveira, Zvonimir Rakamaric, Ganesh Gopalakrishnan. Systematic debugging methods for large-scale HPC computational frameworks // Computing in Science and Engineering. 2014. Vol. 16, no. 3. P. 48–56.

[7] Minh Ngoc Dinh, David Abramso1, Chao Jin, Andrew Gontarek, Bob Moench, Luiz DeRose. A data-centric framework for debugging highly parallel applications // Software - Practice and Experience. 2013. Vol. 45, no. 4. P. 501–526.

[8] Dylan Chapp, Nigel Tan, Sanjukta Bhowmick, Michela Taufer. Identifying Degree and Sources of Non-Determinism in MPI Applications via Graph Kernels // IEEE Transactions on Parallel and Distributed Systems. 2021. Vol. 32, no. 12. P. 2936–2952.

[9] Miguel Pignatelli. TnT: a set of libraries for visualizing trees and track-based annotations for the web // Bioinformatics. 2016. Vol. 32, no. 16. P. 2524–2525.

[10] Cantrell K., Fedarko M.W., Rahman G., McDonald D., Y.,Zaw T., Gonzalez A., Janssen S., Estaki M., Haiminen N., Beck K.L., Zhu Q. Empress enables tree-guided, interactive, and exploratory analyses of multi-omic data sets // mSystems. 2021. Vol. 6, no. 2.

[11] Jadeja M., Kanakia H., Muthu R. Interactive Labelled Object Treemap: Visualization Tool for Multiple Hierarchies // Advances in Intelligent Systems and Computing. 2020. P. 499–508.

[12] Esaie Kuitche, Yanchun Qi, Nadia Tahiri, Jack Parmer,Aïda Ouangraoua . DoubleRecViz: A Web-Based Tool for Visualizing Transcript-Gene-Species tree reconciliation // Bioinformatics. 2021. Vol. 37, no. 13. P. 1920–1922.

[13] Соколов А.П., Першин А.Ю. Графоориентированный программный каркас для реализации сложных вычислительных методов // Программирование. 2019. Т. 47, № 5. С. 43–55.

[14] Соколов А.П. Описание формата данных aDOT (advanced DOT) [Электронный ресурс]. Облачный сервис SA2 Systems. [Офиц. сайт]. 2020. (дата обращения 05.03.2020)

[15] Соколов А.П., Першин А.Ю. Патент на изобретение RU 2681408. Способ и система графо-ориентированного создания масштабируемых и сопровождаемых программных реализаций сложных вычислительных методов. 2019. заявка № RU 2017 122 058 A, приоритет 22.07.2017, опубликовано 22.02.2019

[16] Davidrajuh R. Detecting Existence of Cycles in Petri Nets // Department of Electrical and Computer Engineering. 2016. Vol. 10, no. 12. P. 2936–2951.

[17] Fahad Mahdi, Maytham Safar, Khaled Mahdi. Detecting Cycles in Graphs Using Parallel Capabilities of GPU // Computer Engineering Department. 2011. Vol. 13, no. 12. P. 2936–2952.

[18] Diestel R. Graph Theory: Springer Graduate Text GTM 173. Springer Graduate Texts in Mathematics (GTM). Springer New York, 2012.

[19] ДАТАДВАНС | Программное обеспечение для анализа данных и оптимизации [Электронный ресурс] [Офиц. сайт]. 2021. (дата обращения 06.11.2021).

[20] Расчётные схемы - Руководство пользователя pSeven 6.27 [Электронный ресурс] [Офиц. сайт]. 2021. Дата обращения: 15.11.2021.

[21] Параллельные вычисления - Руководство пользователя pSeven 6.27 [Электронный ресурс] [Офиц. сайт]. 2021. (дата обращения 15.11.2021).