**Разрешение циклических зависимостей графовой модели взаимосвязи требований к программному обеспечению**

**Аннотация**

Получение сведений о трассируемости требований к программному обеспечению (ПО) на файлы исходного кода приложения является сложной прикладной проблемой. Особенно при установлении необходимого объема верификационных процедур, выполнение которых необходимо для подтверждения выполнимости требований к ПО при изменении одного или нескольких файлов исходного кода. Цель работы – это разработка модели, которая бы обеспечивала пользователя информацией о связности файлов исходного кода между собой и требованиями к ПО. Цель достигнута за счет разрешения циклических зависимостей и формирования графа трассируемости требований к ПО на файлы исходного кода, неразрывно связанных друг с другом. Предложена программная реализация модели, в состав которой включены опциональные модули для исследования ее оптимальной конфигурации. Исследована зависимость времени работы от реализации способа хранения данных в разработанном программном решении. Практическая значимость заключается в том, что разработанная модель позволяет получать полный перечень требований к ПО, корректность выполнения которых необходимо проверить после внесения изменений в файлы исходного кода разрабатываемого приложения. В дальнейших исследованиях предполагается сравнение возможностей разработанной модели и метода факторного.

Ключевые слова - требования, трассируемость, исходный код, графовые модели, ориентированный граф, циклы.

**Введение**

Трассируемость требований на файлы исходного кода ПО - это взаимосвязь между требованиями к ПО и файлами исходного кода, которая определяет происхождения, порождения или зависимости между ними. Получение сведений о такой трассируемости является сложной прикладной проблемой. Зачастую такие сведения необходимы для анализа эффективности проведенных работ по проектированию приложения, оценке стоимости его тестирования [3], а также для управления его конфигурацией [1]. В данной работе предложен способ построения схемы, описывающей в каких файлах исходного кода реализуется заданный объем требований к ПО и адаптированной для получения сведений о зависимостях файлов исходного кода между собой.

Для решения задач, связанных с оптимизацией ПО [19], получением данных о загруженности ресурсов вычислительной системы [9], иерархическим описанием зависимостей компонентов [12] широкое применение нашли графовые модели. Это обусловлено простотой описания актуальных для программирования проблем с использованием графовых моделей [14].

При описании работы потоков управления, потоков данных, связи составных частей распределенной системы очень важно указывать направление потока или связи [7], [8]. При организации связей всегда есть источник и есть потребитель. Иногда один компонент системы, обозначаемый на графе вершиной, является одновременно источником и потребителем. Кроме того, он может быть источником для нескольких потребителей и потребителем от нескольких источников. В ряде задач одна вершина может быть источником и потребителем для самого себя, например в случае наличия обратных связей в описываемой модели. Для описания таких связей используются ориентированные графы, в которых по направлению дуг можно судить о принадлежности вершины к числу источников или приемников.

Для решения задач, связанных с оптимизацией многопоточного ПО [5], необходим механизм преобразования графа из исходного вида к целевому. Преобразование происходит по заранее сформулированным правилам и может осуществляться за несколько итераций. Условие окончания проведения итераций преобразования так же определено заранее. Для решения задачи построения графа трассируемости требований к ПО на файлы исходного необходимо, чтобы вне зависимости от очередности вершин графа, к которым применяются действия по преобразованию, результирующий граф всегда формировался бы одинаково. Описанный в статье [5] способ не гарантирует этого.

Изложенный в работе [21] подход к формированию графа с применением алгоритмов нейронных сетей [13] не учитывает ограничение, что вершины результирующего графа должны быть двух категорий. В то же время приведенный способ формирования результирующего графа может быть доработан для решения задачи построения оптимальной декомпозиции компонентов приложения с целью максимизации вариантов комплектаций его поставки при заданном трассировании требований к ПО на файлы исходного кода с учетом разрешенных циклических зависимостей и выполняя свою работу на уже предварительно преобразованном графе.

Описанные в [5] и [17] приемы преобразования ориентированного графа и видоизменения его в результате итерационно выполняемых действий нацелены на построение такого графа, который бы упрощал поиск цепочки задействованных в одном сценарии работы ПО вершин графа. Такие приемы требуют доработки и адаптации для решения задачи построения графа трассируемости требований к ПО на файлы исходного кода с учетом возможности наличия циклических зависимостей у файлов исходного кода между собой.

Кроме того, при описании способов преобразования исходного графа необходимо учитывать временные издержки, которые появляются в ходе выполнения операций над исходным графом. Объем издержек возрастает, если в исходном графе присутствует значительное число требований к ПО и файлов исходного кода.

**Описание модели**

Исходными данными для модели являются: информация о составе файлов исходного кода, сведения об их зависимостях между друг другом, а так же трассируемость требований на них.

Перед началом работы модели необходимо определить:

1. множество файлов исходного кода ;
2. множество требований к ПО ;
3. матрицу бинарных отношений трассируемости требований к ПО на файлы исходного кода размерности ;
4. матрицу бинарных отношений зависимостей файлов исходного кода размерности .

Значение элемента матрицы определяется следующим образом:

Значение элемента матрицы определяется следующим образом:

Примеры исходных матриц и приведены в таблицах и Таблица 1 и Таблица 2 соответственно.

Таблица 1 – Пример матрицы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | … |  |
|  | 1 | 0 | … | 0 |
|  | 0 | 1 | … | 0 |
|  | 0 | 1 | … | 0 |
| … | … | … | … | … |
|  | 0 | 0 | … | 1 |

Таблица 2 – Пример матрицы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  | … |  |
|  | 0 | 0 | 1 | … | 0 |
|  | 0 | 0 | 0 | … | 0 |
|  | 1 | 0 | 0 | … | 0 |
| … | … | … | … | … | … |
|  | 0 | 0 | 0 | … | 0 |

Модель предполагает выполнение следующих этапов:

1. считывание , , и ;
2. построение исходного графа ;
3. преобразование исходного графа в результирующий .

Следуя [10], представляет собой модифицированное И/ИЛИ-дерево, которое помимо стандартных взаимоотношений между структурными элементами по типам И/ИЛИ включает в себя также:

1. отношение множественного ИЛИ;
2. отношение обязательного включения дочернего компонента;
3. отношение опционального включения дочернего компонента;
4. отношения, связывающие между собой компоненты, не являющиеся по отношению друг к другу родительскими или дочерними.

Кроме того, в графе выделяется два слоя:

1. слой файлов исходного кода;
2. слой требований к ПО.

Пример , построенного по входным данным, приведен на рисунке Рисунок 1.

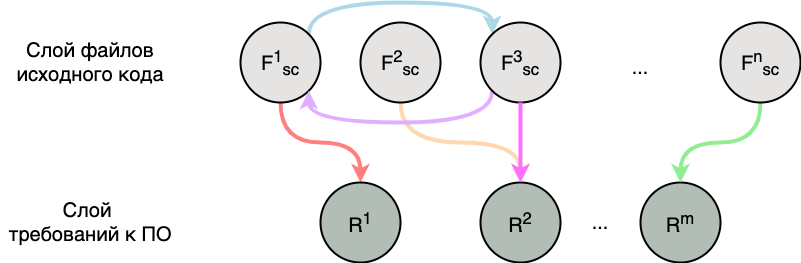


Рисунок 1 – Пример

Для преобразования в итерационно выполняется коррекция [11] до тех пор, пока в присутствуют циклы из произвольного числа узлов [16]. Коррекция выполняется методом слияния [9] входящих в цикл вершин, образуя группу. При этом:

1. дуги между сливающимися вершинами удаляются;
2. образованная группа является новой вершиной графа, все входящие и исходящие дуги замыкаются на нее. Пример слияния приведен на рисунке Рисунок 2.

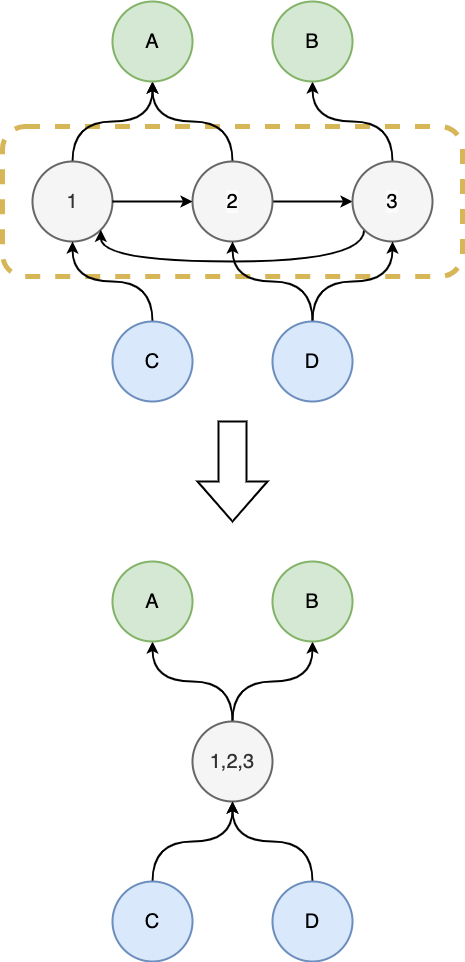


Рисунок 2 – Пример слияния входящих в цикл вершин

На последующей итерации поиска циклов в графе обозначенные выше группы учитываются как обычные вершины и, таким образом, могут участвовать в образовании новых групп по вышеуказанным правилам.

В приведенном примере (см. рис. Рисунок 1) вершины и образуют цикл. Результат слияния приведен на рисунке Рисунок 3.

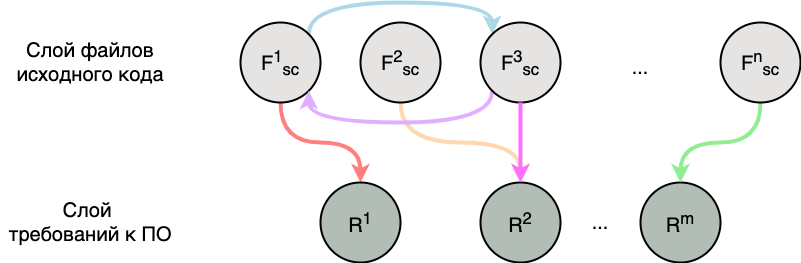


Рисунок 3 – Пример

Результирующий граф может быть использован при оценке объема верификационных процедур, которые необходимо выполнить при внесении изменений в файлы исходного кода. Так, при внесении изменений в файл исходного кода следует выполнить верификационные процедуры реализации требований и .

**Программная реализация**

Описанная модель реализована на языке программирования Java. Реализация сочетает достоинства применения как парадигмы ООП, так и функционального программирования за счет активного использования лямбда-выражений.

При проектировании программной реализации было выявлено, что на производительность решения существенное влияние оказывает способ хранения информации о вершинах и дугах. Вариативность способов хранения этой информации зависит от выбранного языка программирования и объема подключаемых библиотек.

Используемый в работах [12], [18], [20] способ хранения информации в массивах является самым простым в реализации. Его использование позволяет наиболее эффективно осуществлять поиск элементов и осуществлять обновление информации о наличествующих в графе вершинах и дугах. Например, как показано в работе [15], изменять значение весовых коэффициентов. Данный способ наиболее эффективно использовать при работе на статическом графе, который не изменяется с течением времени. Использование его в динамически изменяющимся графе приведет к необходимости создания новых экземпляров массивов с последующим заполнением их значений.

В работе [4] приводится пример использования многомерных массивов. Однако, как описано ранее, данный подход эффективен при неизменяемом числе связей между компонентами графа. Например, если у каждой вершины зафиксировано число дуг, а в процессе преобразований графа моделью изменяется информация о том, в какие именно дуги соединяют вершину с другими вершинами графа. При проектировании модели, в которой число дуг у вершины графа может изменяться, могут создаваться новые вершины и дуги, удаляться ранее созданные, эффективным представляется использование динамических структур данных.

Примером динамических структур данных являются связные списки. В работе [6] рассматривается пример их использования. Однако списки допускают хранение одинаковых элементов, что при проектировании описанной графовой модели недопустимо. Модель предусматривает наличие уникальных элементов. Вследствие этого было принято решение об использовании множеств уникальных элементов. В Java такая структура данных носит название Set.

Выбранная структура данных представляет коллекцию, которая может быть реализована по-разному. Выбор реализации влияет на время работы программной реализации модели. С целью проведения экспериментов и получения информации о зависимости времени работы программы от числа элементов графа в реализации коллекции Set, предусмотрены два модуля для взаимодействия с данными, заданных пользователем:

1. модуль определения реализации коллекции Set;
2. модуль генерации исходных данных.

Модуль определения реализации коллекции Set выполнен для оптимальности применения различных реализаций коллекции Set в разработанной программной реализации. Благодаря использованию этого модуля достигается возможность задания конкретной реализации с последующим ее внедрением в исходный код программной реализации модели. Модуль позволяет внедрять как стандартные реализации коллекции Set, так и реализации, хранящиеся в подключаемых библиотеках или самостоятельно реализованных пользователем.

Модуль генерации исходных данных выполнен для сокращения времени на подготовку и .

В качестве входных данных модуль принимает четыре параметра:

1. - число файлов исходного кода;
2. - число требований к ПО;
3. - максимальное число зависимостей у файла исходного кода;
4. - максимальное число реализуемых требований в одном файле.

Диаграмма работы выполненной программной реализации приведена на рисунке Рисунок 4.

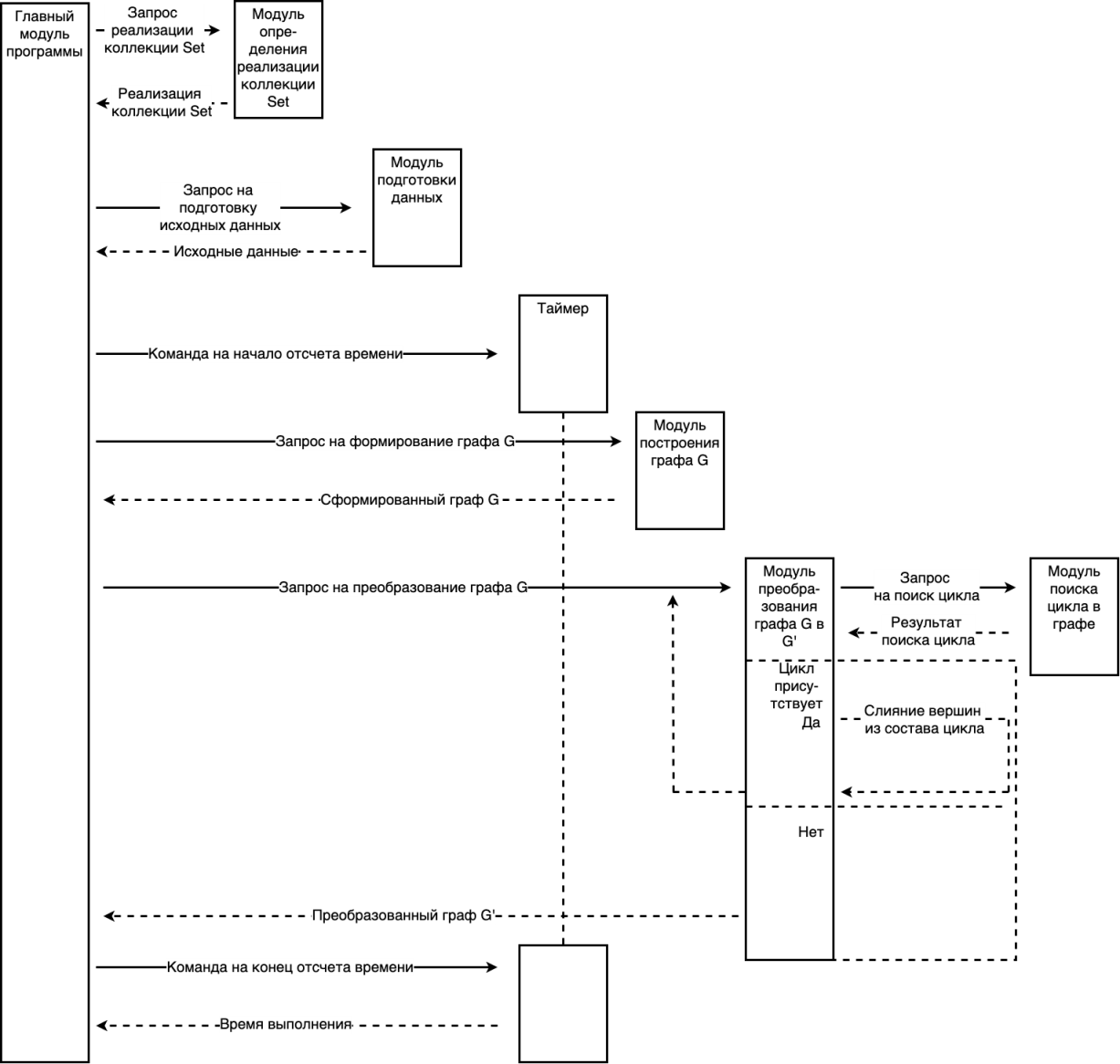


Рисунок 4 – Диаграмма работы выполненной программной реализации

**Эксперименты и обсуждение**

В целях проверки работоспособности программной реализации описанной модели проводилась серия экспериментов. В качестве входных данных использовались различные комбинации значений параметров для модуля генерации исходных данных.

Характеристики оборудования, используемого для проведения экспериментов:

1. операционная система - Ubuntu 23.04;
2. процессор - 2‑ядерный процессор Intel Core i5 с тактовой частотой 1,8 GHz;
3. объем ОЗУ - 8 ГБ.

В таблицах Таблица 3 - Таблица 5 приведены сведения о среднем времени выполнения операций в экспериментах по преобразованию в , которые были выполнены по 1000 раз каждый при заданных параметрах генератора исходных данных для следующих стандартных реализаций коллекции Set:

1. HashSet – таблица Таблица 3;
2. LinkedHashSet – таблица Таблица 4;
3. TreeSet – таблица Таблица 5.

На рисунках Рисунок 5 и Рис. **6** приводится графическая интерпретация результатов экспериментов при работе с числом файлов исходного кода в 100 и 1000 единиц соответственно.

Таблица 3 – Результаты работы программной реализации с применением HashSet

| **№** | **Fc** | **Rc** | **Mfd** | **Mfr** | **Среднее время, мс** |
| --- | --- | --- | --- | --- | --- |
| 1 | 100 | 100 | 30 | 30 | 42 |
| 2 | 100 | 100 | 30 | 50 | 52 |
| 3 | 100 | 100 | 50 | 30 | 60 |
| 4 | 100 | 100 | 50 | 50 | 41 |
| 5 | 100 | 1000 | 30 | 30 | 57 |
| 6 | 100 | 1000 | 30 | 50 | 77 |
| 7 | 100 | 1000 | 50 | 30 | 63 |
| 8 | 100 | 1000 | 50 | 50 | 65 |
| 9 | 1000 | 100 | 30 | 30 | 7830 |
| 10 | 1000 | 100 | 30 | 50 | 7402 |
| 11 | 1000 | 100 | 50 | 30 | 8364 |
| 12 | 1000 | 100 | 50 | 50 | 8163 |
| 13 | 1000 | 1000 | 30 | 30 | 6932 |
| 14 | 1000 | 1000 | 30 | 50 | 6807 |
| 15 | 1000 | 1000 | 50 | 30 | 8197 |
| 16 | 1000 | 1000 | 50 | 50 | 8496 |

Таблица 4 – Результаты работы программной реализации с применением LinkedHashSet

| **№** | **Fc** | **Rc** | **Mfd** | **Mfr** | **Среднее время, мс** |
| --- | --- | --- | --- | --- | --- |
| 1 | 100 | 100 | 30 | 30 | 37 |
| 2 | 100 | 100 | 30 | 50 | 52 |
| 3 | 100 | 100 | 50 | 30 | 40 |
| 4 | 100 | 100 | 50 | 50 | 29 |
| 5 | 100 | 1000 | 30 | 30 | 88 |
| 6 | 100 | 1000 | 30 | 50 | 48 |
| 7 | 100 | 1000 | 50 | 30 | 52 |
| 8 | 100 | 1000 | 50 | 50 | 71 |
| 9 | 1000 | 100 | 30 | 30 | 4061 |
| 10 | 1000 | 100 | 30 | 50 | 2894 |
| 11 | 1000 | 100 | 50 | 30 | 3065 |
| 12 | 1000 | 100 | 50 | 50 | 3083 |
| 13 | 1000 | 1000 | 30 | 30 | 2755 |
| 14 | 1000 | 1000 | 30 | 50 | 2892 |
| 15 | 1000 | 1000 | 50 | 30 | 3017 |
| 16 | 1000 | 1000 | 50 | 50 | 3415 |

Таблица 5 – Результаты работы программной реализации с применением TreeSet

| **№** | **Fc** | **Rc** | **Mfd** | **Mfr** | **Среднее время, мс** |
| --- | --- | --- | --- | --- | --- |
| 1 | 100 | 100 | 30 | 30 | 65 |
| 2 | 100 | 100 | 30 | 50 | 71 |
| 3 | 100 | 100 | 50 | 30 | 70 |
| 4 | 100 | 100 | 50 | 50 | 52 |
| 5 | 100 | 1000 | 30 | 30 | 51 |
| 6 | 100 | 1000 | 30 | 50 | 58 |
| 7 | 100 | 1000 | 50 | 30 | 78 |
| 8 | 100 | 1000 | 50 | 50 | 119 |
| 9 | 1000 | 100 | 30 | 30 | 8083 |
| 10 | 1000 | 100 | 30 | 50 | 6264 |
| 11 | 1000 | 100 | 50 | 30 | 7509 |
| 12 | 1000 | 100 | 50 | 50 | 7562 |
| 13 | 1000 | 1000 | 30 | 30 | 6640 |
| 14 | 1000 | 1000 | 30 | 50 | 6911 |
| 15 | 1000 | 1000 | 50 | 30 | 7982 |
| 16 | 1000 | 1000 | 50 | 50 | 7705 |

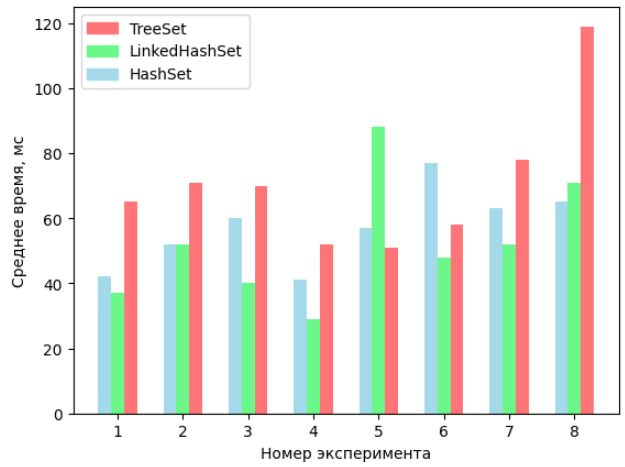


Рис. 5 - Графическая интерпретация результатов экспериментов при работе с числом файлов исходного кода в 100 единиц

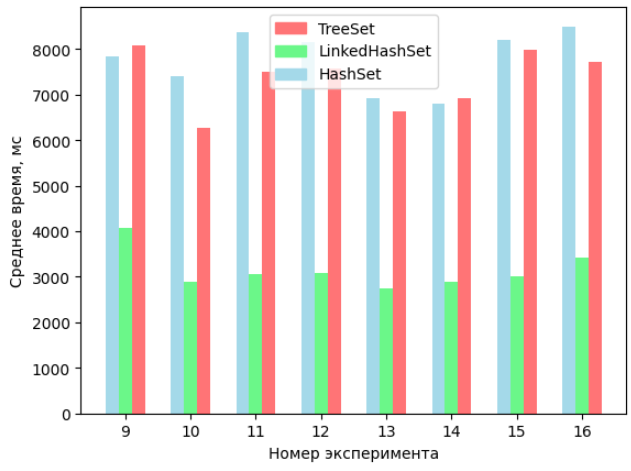


Рис. 6 - Графическая интерпретация результатов экспериментов при работе с числом файлов исходного кода в 1000 единиц

Результаты экспериментов показывают, что резкое увеличение времени выполнения преобразования графа происходит при увеличении числа файлов исходного кода, а значительное увеличение числа требований к ПО к такому эффекту не приводит.

Наибольшее значение среднего времени составляет 8496 мс, оно достигается при работе с реализацией HashSet. Наименьшее значение среднего времени достигается при работе с LinkedHashSet. При использовании этой реализации коллекции Set наибольшее значение среднего времени составляет 4061 мс, в то время как для TreeSet максимальное значение среднего времени достигает 8083 мс в экспериментах по взаимодействию с файлами исходного кода в количестве 1000 единиц.

**Заключение**

В статье предложена модель, отображающая трассируемость требований к ПО на файлы исходного кода и позволяющая разрешать циклические зависимости между файлами исходного кода. Используя ее можно получать информацию о степени связности файлов исходного кода друг с другом, на какую долю функционала оказывается влияние при внесении изменений в файлы исходного кода. Данные сведения могут быть полезны при оценке необходимого объема выполнения верификационных процедур после внесения изменений в один или несколько файлов исходного кода.

В дальнейших исследованиях предполагается проведение сравнения предложенного подхода с методом факторного моделирования [2] как альтернативного представления вычислительных процедур на графах, без устранения циклов и позволяющих вычислять взаимное влияние вершин.

**Библиографический список**

1. В. П. Осипов, В. А. Судаков, Г. Ф. Хахулин. Информационные технологии формирования этапной программы научно-прикладных исследований на российском сегменте Международной космической станции // Вестник компьютерных и информационных технологий. – 2012. – № 12(102). – С. 24-28.
2. Четверушкин, Б. Н. Факторное моделирование для инновационно-активных предприятий / Б. Н. Четверушкин, В. А. Судаков // Математическое моделирование. – 2020. – Т. 32, № 3. – С. 115-126. – DOI 10.20948/mm-2020-03-07. – EDN PBCNYP.
3. А.Н. Вигура, анализ и тестирование программ на основе алгебраической модели, Информационные технологии, Вестник Нижегородского университета им. Н.И. Лобачевского, 2011, No 5 (1), с. 185–190;
4. А.М. Шульженко, автоматическое определение циклов ParDo в программе, Естественные науки, известия ВУЗов. северо-кавказский регион, ISSN 0321-3005, с. 77-87;
5. B. П. Корячко, д-р техн. наук проф., C. В. Скворцов, канд. техн. наук доц., Иерархическая модель глобальной оптимизации у параллельных объектных программ, электронный журнал "Инженерное образование", 2006;
6. Кошелев В.К., Игнатьев В.Н., Борзилов А.И. Инфраструктура статического анализа программ на языке C\#. Труды ИСП РАН, том 28, вып. 1, 2016 г., с. 21-40;
7. А. А. Чертков, Я. Н. Каск, Л. Б. Очина, Маршрутизация потоковой сети на основе модификации алгоритма Беллмана - Форда, ФГБОУ ВО «ГУМРФ имени адмирала С. О. Макарова», Санкт-Петербург, Российская Федерация, 2022г, топ 14 № 4, с 615-627;
8. В. В. Сахаров, А. А. Чертков, Л. Б. Очина, Маршрутизация сетей с отрицательными весами звеньев в пакете оптимизации MATLAB ФГБОУ ВО «ГУМРФ имени адмирала С. О. Макарова», Санкт-Петербург, Российская Федерация, 2019г, том 11 № 2, с 230-242;
9. К.В. Недоводеев, Метод генерации графов потоков данных, используемых при автоматическом синтезе параллельных программ для неоднородных многоядерных процессов, Научно-технические ведомости СПбГПУ 3' 20122 Информатика. Телекоммуникации. Управление, с 47-52;
10. Ю. И. Евсеева, А. C. Бождай, Метод структурно-параметрического синтеза адаптивных программных компонентов виртуальной образовательной среды, Известия высших учебных заведений. Поволжский регион, DOI 10.21685/2072-3059-2016-3-8, с 84-92;
11. О.А. Четверина, Методы коррекции профильной информации в процессе компиляции, Труды ИСП РАН, том 27, вып. 6, 2015 г. с 49-65;
12. О.Б. Штейнберг, Минимизация количества временных массивов в задаче разбиения циклов, ISSN 0321-3005 известия ВУЗов, Северо-Кавказский регион, естественные науки, 2011. № 5, с 31-35;
13. Тарков М. С., Об эффективности построения гамильтоновых циклов в графах распределенных вычислительных систем рекуррентными нейронными сетями, Информационные технологии в управлении, Институт физики полупроводников им. А.В. Ржанова СО РАН, Новосибирск, Управление большими системами. Выпуск 43, 2013, с 157-171;
14. С.В. Огородов, Обоснование линейноупорядоченного представления графовых моделей программ, Институт «Кибернетический центр» ТПУ, Известия Томского политехнического университета. 2008. Т. 312. № 5, с 85-89;
15. Фролов А. С., канд. техн. наук Семенов А. С, Обзор проблемно-ориентированных языков программирования для параллельного анализа статических графов, Computational nanotechnology 1-2017, ISSN 2313-223X, 27-32;
16. Е. П. Емельченков, В. И. Мунерман, Д. В. Мунерман, Т. А. Самойлова, Один метод построения циклов в графе, Современные информационные технологии и ИТ-образование. 2021. Т. 17, № 4. С. 814-823;
17. А.А. Каленкова, Оптимизация потоков работ по времени выполнения, основанная на удалении избыточных потоков управления, труды МФТИ. — 2009. — Том 1, № 2, 160-174;
18. А.П. Баглий, Н.М. Кривошеев, Б.Я. Штейнберг, О.Б. Штейнберг, Преобразования программ в Оптимизирующей распараллеливающей системе для распараллеливания на распределенную память, Инженерный вестник Дона, №12 (2022), ivdon.ru/ru/magazine/archive/n12y20225/8089;
19. О.Б. Штейнберг, И.А. Ивлев, Применение преобразования циклов "Retiming" с целью уменьшения количества используемых регистров, Южный федеральный университет, г. Ростов-на-Дону, Россия, ISSN 0321-2653 известия ВУЗов. Северо-Кавказский регион. Технические науки. 2017. № 3, с 76-80;
20. А. Ю. Попов, Применение вычислительных систем смногими потоками команд и одним потоком данных для решения задач оптимизации, ISSN 0236-3933. Вестник МГТУ им. Н.Э. Баумана. Сер. "Приборостроение", 2012, с 146-154;
21. Карпов Ю.Л., Волкова И.А., Вылиток А.А., Карпов Л.Е., Сметанин Ю.Г. Проектирование интерфейсов классов графовой модели нейронной сети. Труды ИСП РАН, том 31, вып. 4, 2019 г., стр. 97-112.