Разрешение циклических зависимостей графовой математической модели триссируемости требований к ПО на файлы исходного кода

Аннотация

Некоторый текст анотации

Введение

Получение сведений о трассируемости требований к программному обеспечению (ПО) на файлы исходного кода приложения является сложной прикладной проблемой. Зачастую такие сведения необходимы для анализа эффективности проведенных работ по проектированию приложения, оценке стоимости его тестирования, а так же управления его конфигурацией. В данной работе описывается 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, получением данных о загруженности ресурсов вычислительной системы, иерархическим описанием зависимостей компонентов, получением данных о вызове процедур широкое применение нашли графовые математические модели. Это обусловлено простотой описания актуальных для программирования проблем в терминах графов, а значит они могут быть решены при помощи идентичных хорошо изученных и математически обоснованных механизмов из теории графов.

При описании работы потоков управления, потоков данных, связи составных частей распределенной системы очень важно указывать направление потока или связи [5], [6]. 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 один узел 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 многопотоного ПО [3] 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 [3] способ не гарантирует этого.

Изложенный в работе [20] подход к формированию графа с применением алгоритмов нейронных сетей не учитывает ограничение, что вершины результирующего графа должны быть 2х 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 [3] и [16] 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 треований 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 при выполнении операций над исходным графом. Объем издержек возрастает при наличии в исходном графе значительного числа требований к ПО и файлов исходного кода.

Описание модели

Разрешение циклических зависимостей графовой математической модели трисируемости требований к ПО на файлы исходного кода проидит в несколько этапов:

1. считавание исходных данных;
2. построение первичного графа трассируемости требований к ПО на файлы исходного кода приложения;
3. пока в графе на слое файлов исходного кода присутствуют циклы выполнятся слияние входящих в цикл вершин;
4. формирование результирующего графа трассируемости.

Слияние входящих в цикл вершин следует производить итерационно до тех пор пока в результирующем графе не будут отсутствовать циклы. После объединения вершин в одну группу граф видоизменяется. При этом для дуг, которые объединяются выполняется:

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

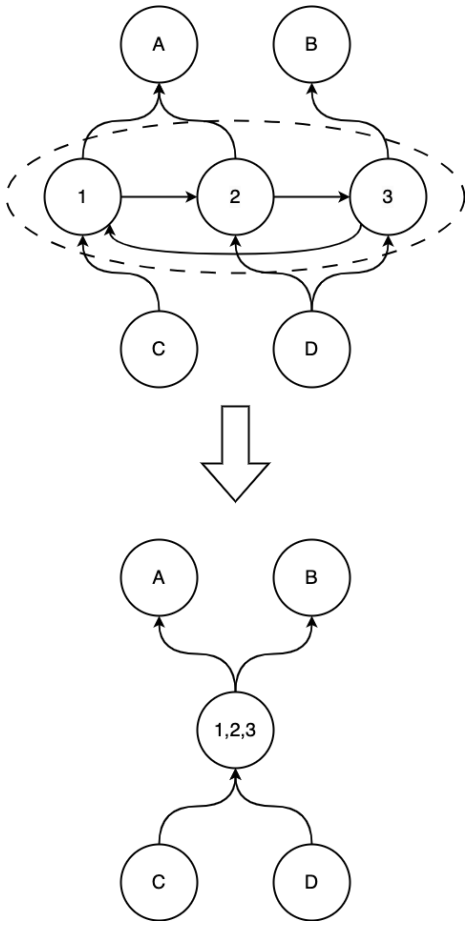

Рис. 1: Пример слияния входящих в цикл вершин

На следующей итерации поиска циклов в графе вышеобозначенные группы учитываются как обычные вершины и таким образом могут учавствовать в образовании новых групп по вышеуказанным правилам.

Перед началом работы модели необходимо подготовить данные: все требования к ПО и файлы исходного кода программы должны быть проидентифицированы натуральными числами по возрастанию начиная с 1. В приводимых примерах литерой m обозначется максимальный идентификатор требования к ПО, литерой n обозначается максимальный идентификатор файла исходного кода.

Исходными данными для модели являются две матрицы бинарных отношений:

1. матрица трассируемости требований на файлы исходного кода. В ней индекс столбца матрицы соответствует идентификатору требования, индекс строки - идентификатору файла исходного кода. Если файл исходного кода реализует требование, значение соответствующей элемента матрицы устанавливается равным 1, иначе 0 (рисунок 2);

2. матрица зависимостей между файлами исходного кода. В ней индексы столбцов и строк соответствуют идентификаторам файлов исходного кода. Значение элемента матрицы устанавливается равным 1 если файл исходного кода с идентификатором номера строки имеет зависимость на файл исходного кода с идентификатором номера столбца, иначе значение элемента устанавливается равным 0. В рамках решаемой задачи необходимость отображать зависимости файла исходного кода на самого себя отсутствует. В следствие этого элементы матрицы на главной диагонале должны быть равны 0 (рисунок 3).

	Треб 1	Треб 2	...	Треб m
Файл 1	1	0	...	0
Файл 2	0	1	...	0
Файл 3	0	1	...	0
...	...	...	...	...
Файл n	0	0	...	1

Рис. 2: Пример матрицы трассируемости требований на файлы исходного кода

	Файл 1	Файл 2	Файл 3	...	Файл n
Файл 1	0	0	1	...	0
Файл 2	0	0	0	...	0
Файл 3	1	0	0	...	0
...	...	...	...	...	...
Файл n	0	0	0	...	0

Рис. 3: Пример матрицы зависимостей файлов исходного кода друг на друга

По входным данным строится изначальный граф. В изнчальном графе выделяются два слоя:

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

Пример изначального графа, построенного по входным данным приведен на рисунке 4.

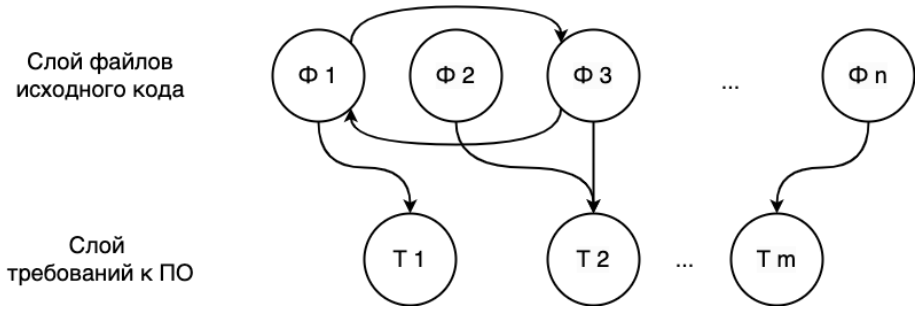

Рис. 4: Пример изначального графа

В приведенном примере вершины 1 и 3 образуют цикл. Результат слияния приведен на рисунке 5.

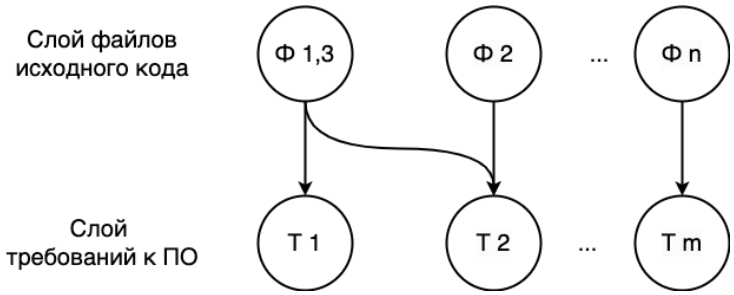

Рис. 5: Пример результирующего графа

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 с идентификатором 1 требуется выполнить верификационные процедуры реализации требований с идентификаторами 1 и 2. Для этого необходимо выполнить тестовые процедуры относящиеся к файлам исходного кода с идентификаторами 1, 2 и 3.

Программная реализация

При достаточно большом количестве элементов изначального графа трассируемости требований на файлы исходного кода не представляется возможным его преобразование и последующая оценка эффективности без применения инструментов предоставляемых вычислительной техникой и возможностей языков программирования. Не существует универсального способа описания графа на языке программирования с целью применения полученной реализации для решения произвольной задачи. В каждом отдельном случае, для каждой отдельно взятой задачи существует необходимость описания графовой модели и алгоритмов

действий над ней. Однако существуют распространенные практики, позволяющие эффективно описывать графовые математические модели на тех или иных языках программирования.

Так, в работах ... для описания состава узлов и связей между ними применяются массивы целочисленных данных. В работах ... продемонстрировано использование динамических списков. Широкое применение для решения такого класса задач нашли структуры, их возможности описаны в ... .

Эксперименты и обсуждение

Заключение