УДК 004.02+519.1

А.В. Погребной

Выделение общей части в версиях исходного кода программ, представленного графовой моделью

Решение задач анализа структурного сходства объектов на основе применения графовых моделей (ГМ) является одним из актуальных направлений исследований в прикладной теории графов. В статье в качестве объектов для анализа сходства рассматриваются версии исходного кода программ. Существующие системы управления версиями (СУВ) при решении данной задачи работают непосредственно с текстами исходных кодов версий программ. Наделяя СУВ возможностью работать с ГМ, адекватно представляющими исходный код версий программ, позволяет существенно улучшить качество выполнения традиционных функций СУВ и расширить инструментарий по сопровождению процесса разработки программ. Оценивать сходство версий программ предлагается на основе выделения в них наибольшей общей части, т.е. наибольшего общего подграфа. Цель исследований заключается в разработке метода решения задачи выделения наибольшей общей части в версиях программ, представленных в виде ГМ. Исследования базируются на применении предложенной автором теории дифференциации вершин графа, в которой модель структуры графа представляется в виде сети автоматов. Важным преимуществом метода является его способность легко адаптироваться к особенностям написания исходного кода программы и используемой версии языка программирования. Метод включает три этапа – составление и кодирование списков атрибутов, приписанных вершинам и дугам ГМ; преобразование ГМ в стандартную форму в виде ориентированного графа; выделение общей части в двух ориентированных графах, представляющих сравниваемые версии программы. Алгоритм выделения общей части, работая со стандартными формами представления ГМ, является инвариантным относительно особенностей описания ГМ и набора используемых атрибутов. Приводится пример решения задачи выделения общей части для двух сравниваемых фрагментов ГМ.

Ключевые слова: графовая модель программы, система управления версиями, общая часть двух графов, подстановка сходства графов, дифференциация вершин графов.

doi: 10.21293/1818-0442-2018-21-3-49-56

Разработка программной системы, как правило, связывается с созданием последовательности работоспособных версий [1–3]. За время разработки над проектом обычно работает множество человек, которые постоянно вносят изменения в исходный код программы. Таким образом, накапливается множество версий исходного кода программы, с которыми невозможно работать без специальных инструментов.

Для работы с версиями программ применяются системы управления версиями (СУВ) [4, 5], например: Git, Mercurial [6, 7]. Данные СУВ работают непосредственно с текстами исходных кодов версий программ. Это порождает ряд проблем, связанных с неоднозначностью форматирования текстов исходного кода, а также невозможностью анализа изменений с целью более интеллектуального разрешения возникающих конфликтов.

Исходный код программ для различных целей представляется в виде самых разных графовых моделей (ГМ) [8–10]. В [11] для поддержки процесса разработки и анализа программ предложено исходный код представлять в виде ГМ, полностью описывающей исходный код программы, т.е. позволяющей конвертировать графовую модель обратно в исходный код без потерь какой-либо информации.

В предлагаемой ГМ вершинами являются фрагменты синтаксических конструкций языка, такие как объявления переменных, вызовов операторов или функций. Соответственно для разных языков программирования правила построения такого графа будут различаться. На рис. 1 приведён пример ГМ для фрагмента исходного кода на языке Java.

В примере для удобства визуализации все вершины разделены на два типа – данные и модули (некоторые действия). Данные в ГМ (замкнутые кривые) имеют три вида изображения. Сплошная линия соответствует локальной переменной. Сплошная линия с точкой отражает свойство класса. Пунктиром изображается промежуточная переменная без указания имени.

Модули изображаются прямоугольниками двух видов. Прямоугольник без точки соответствует модулю, выполняющему оператор, метод. Прямоугольник с точкой обозначает объявление метода класса.

Информационные связи изображаются с указанием стрелок. В тех случаях, когда информационные входы в модуль необходимо различать, то у стрелки входной дуги указывается номер входа (например, для цикла `for`). Связь без ориентации применяется для отражения принадлежности модуля или свойства объекту.

Управляющие связи (m_{j1} , m_{j2}) между модулями помечаются точками, которые условно воспринимаются в качестве переменных. Такие переменные соответствуют появлению сигнала о том, что модуль m_{j1} закончил работу, а модуль m_{j2} может приступать к выполнению.

Приведенный на рис. 1 набор элементов визуализации представляет только те синтаксические конструкции языка Java, которые задействованы в приведенном примере. Этот набор для языка в целом содержит и другие элементы, которые здесь не представлены.

```

Hash hash = new Hash(10);
int sum = 0;
for (int i = 0; i < args.length; i++) {
    sum += hash.calc(args[i]);
}

```

```

class Hash {
    private int b;
    Hash(int b) {
        this.b = b;
    }
    int calc(int a) {
        return (a + b) * 3;
    }
}

```

Рис. 1. Пример ГМ для фрагмента исходного кода на языке Java

Таким образом, каждая вершина в графе имеет набор атрибутов, необходимых для адекватного представления исходного кода. Атрибут, который есть у всех вершин, – это тип вершины. Он же определяет, какие атрибуты могут быть у вершины. Например, если тип вершины – объявление переменной, то у неё будет два атрибута: тип переменной и её имя. Также в атрибутах вершин содержится различная информация о форматировании исходного кода вроде отступов, которая позволяет восстановить исходный код по ГМ.

Наряду с задачами анализа функционирования программ построение такой ГМ даёт возможность расширить и повысить эффективность выполнения задач СУВ. Функционирование СУВ, основанной на ГМ, будет включать три этапа:

1. Построение ГМ для исходного кода.
2. Выявление различий в ГМ, представляющих две разных версии исходного кода.
3. Интерпретация и анализ полученных на предыдущем этапе различий.

Решению второго этапа в условиях, когда версии программы представлены в виде ГМ₁ и ГМ₂, посвящена данная статья.

В статье предлагается метод, основанный на преобразовании ГМ в стандартную форму представления в виде ориентированного графа. По этому методу версии ГМ₁ и ГМ₂ преобразуются соответственно в ориентированные графы G_1 и G_2 . Задача выделения общей части в версиях программ в этом случае сводится к задаче определения сходства двух

ориентированных графов G_1 и G_2 . Таким образом, достигается независимость алгоритма от выбранного языка программирования. Обычно при решении задач сходства используются косвенные оценки, как, например, в [12, 13]. В версиях же программ для работы СУВ необходимо определение наибольшей общей части, поэтому задача решается на основе методов, изложенных в [14]. После выделения в графах G_1 и G_2 общей части в виде подграфа Q она может быть представлена отдельной ГМ_Q и при необходимости преобразована в текстовую форму в виде исходного кода.

Метод выделения общей части в графовых моделях версий программ

Представление версий программы в виде ГМ сопровождается указанием большого разнообразия атрибутов, приписываемых вершинам и рёбрам ГМ [11]. Такое представление позволяет на основе структуры ГМ и множества атрибутов восстановить исходный код соответствующей версии программы. Вместе с тем исследования показали, что выделение общих частей в версиях непосредственно на основе ГМ₁ и ГМ₂ оказалось трудно реализовать по ряду причин: громоздкое описание ГМ; отсутствие возможности применения метода взаимозависимой дифференциации вершин для определения сходства структур графов [14]; зависимость алгоритма выделения общей части от версий языка программирования.

Каждая из приведенных выше причин указывает на необходимость разработки иного подхода к решению проблемы выделения общих частей в вер-

сиях программ и его применения в составе СУБ. В основу этого подхода положено предварительное преобразование ГМ в стандартную форму, которая не зависит от состава атрибутов, используемых в ГМ.

В данном подходе все особенности исходного кода, в том числе и для разных языков, должны учитываться на стадии формирования описания ГМ и базовых списков атрибутов. Преобразование ГМ в стандартную форму выполняется по универсальным правилам и независимо от особенностей исходного кода представляет версии ГМ₁ и ГМ₂ ориентированными графами G_1 и G_2 соответственно.

Решение проблемы выделения общей части включает три этапа. На первом этапе на основе описания ГМ формируются новые или уточняются имеющиеся базовые списки атрибутов ($V, X, Y, \dots$) для вершин и (W, B, H) для дуг. Эти списки отражают особенности языка программирования и включают множества атрибутов, используемых при описании ГМ.

Список V включает атрибуты вида вершин (оператор, переменная, метод, константа и т.д.). Списки $X, Y, \dots$ соответствуют идентификаторам вершин, типам вершин и другим атрибутам, входящим в описание вершины. Список W включает атрибуты вида дуг (информационные, управляющие, отражающие отношение принадлежности, зависимости и т.д.). Списки B, H содержат атрибуты, характеризующие входы и выходы дуг.

На втором этапе выполняются операции по преобразованию ГМ₁ и ГМ₂ в G_1 и G_2 . Предварительно базовые списки атрибутов пополняются новыми атрибутами из описаний ГМ₁ и ГМ₂. Далее каждая ГМ представляется четвёркой (E, U, A_E, A_U) , включающей множество вершин E , множество дуг U , множество кодов атрибутов вершин $A_E = \{e_i(V_i, x_i, y_i, \dots)\}$, множество кодов атрибутов дуг $A_U = \{(e_j, e_i)(W_{ji}, b_i, h_j)\}$. Здесь коды $(V_i, x_i, y_i, \dots)$ и (W_{ji}, b_i, h_j) являются порядковыми номерами соответствующих атрибутов вершин и дуг, которые они занимают в базовых списках ($V, X, Y, \dots$) и (W, B, H).

В итоге описания ГМ₁ и ГМ₂ представляются ориентированными графами G_1 и G_2 , у которых атрибуты вершин и дуг заменены на порядковые номера (коды) в соответствующих списках. Структура каждого графа представляется в виде списка инциденторов входных вершин $\{e_i(F(e_i))\}$, где инцидентор $F(e_i)$ включает множество вершин e_j , из которых дуги выходят и поступают на вход в вершину e_i . Коды атрибутов для вершин e_i и дуг (e_j, e_i) представляются множествами записей в виде $\{e_i(V_i, x_i, y_i, \dots)\}$ и $\{(e_j, e_i)(W_{ji}, b_i, h_j)\}$. При отсутствии b_i и h_j указывается только W_{ji} , т.е. запись имеет вид $(e_j, e_i)(W_{ji})$. Если отсутствует b_i или h_j , то записи имеют вид $(e_j, e_i)(W_{ji}, \emptyset, h_j)$ или $(e_j, e_i)(W_{ji}, b_i)$.

Третий этап включает реализацию алгоритма выделения в графах G_1 и G_2 общей части Q . В основу алгоритма положен метод оценивания сходства структур двух графов [11]. Общая часть Q соответ-

ствует подстановке сходства вершин графов G_1 и G_2 , которой соответствует наибольшее число совпадающих дуг. Перед решением задачи выделения общей части в графах G_1 и G_2 удаляются вершины и связанные с ними дуги, которые не могут войти в общую часть. Вершина e_i удаляется из графа G_1 , если запись $e_i(V_i, x_i, y_i, \dots)$ отсутствует среди записей для графа G_2 . Аналогично из графа G_2 вершина удаляется, если её запись не найдена среди записей графа G_1 . Оставшиеся части графов G_1 и G_2 обозначим графами G и H . При этом часть с меньшим числом вершин будем обозначать как G . Графы G и H характеризуются тем, что для каждой записи в графе G есть совпадающая с ней запись в графе H .

Задача выделения общей части в ГМ₁ и ГМ₂ решается на соответствующих графах $G = (E_G, U_G)$ и $H = (E_H, U_H)$. Каждый из них представлен списком инциденторов $\{e_i(F(e_i))\}$ и множествами записей кодов атрибутов вершин $\{e_i(V_i, x_i, y_i, \dots)\}$ и дуг $\{(e_j, e_i)(W_{ji}, b_i, h_j)\}$. Решение основывается на применении метода взаимозависимой дифференциации вершин графов G и H .

Для применения метода определяются начальные состояния моделей структур графов. С этой целью в графе G для каждой i -й записи в множестве $\{e_i(V_i, x_i, y_i, \dots)\}$ назначается код d_i^0 – интегральный описатель кодов атрибутов вершины e_i . Запись с наименьшим числом атрибутов и минимальным значением среди V_i , затем среди x_i и т.д. получает код $d_i^0 = 1$. Все записи, совпадающие с выбранной, также получают код $d_i^0 = 1$. Выбор записей для назначения кода $d_i^0 = 2$ производится среди оставшихся записей аналогично. В итоге все различающиеся записи получают разные коды d_i^0 . В графе H вершины e_j получают код $d_j^0 = d_i^0$, если записи вершин e_i и e_j в графах G и H совпадают.

После назначения вершинам $e_i \in E_G$ кодов d_i^0 , а вершинам $e_j \in E_H$ кодов d_j^0 начальное состояние моделей структур графов G и H представляется векторами $\mathbf{D}_G^0 = \{d_i^0\}$ и $\mathbf{D}_H^0 = \{d_j^0\}$. С учётом инциденторов вершин получим векторы $\mathbf{D}_G^0(F) = \{d_i^0(F(d_i^0))\}$ и $\mathbf{D}_H^0(F) = \{d_j^0(F(d_j^0))\}$. В инциденторах $F(d_i^0)$ и $F(d_j^0)$ помимо кодов входных вершин указываются коды атрибутов соответствующих дуг, например: $F(d_{10}^0) = (d_6^0(W_{6,10}, \emptyset, h_6), d_{14}^0(W_{14,10}), d_{20}^0(W_{20,10}, b_{10}))$.

Сохранение в записях инциденторов кодов атрибутов дуг в виде (W_{ji}, b_i, h_j) позволяет сравнивать и выделять в графах G и H дуги с частично совпадающими атрибутами. Это важно в условиях, когда выделение общей части соответствует поиску подстановки сходства с наибольшим числом полностью или частично совпадающих дуг.

Основные положения метода выделения общей части разработаны с учётом особенностей графов G и H и заключаются в следующем:

1. Векторы $\mathbf{D}_G^0(F)$ и $\mathbf{D}_H^0(F)$ отражают начальную дифференциацию вершин и разбивают их на группы $E_d^0(G)$ и $E_d^0(H)$ независимо от степеней по входным дугам. Из этого следует, что шаги взаимозависимой дифференциации должны предусматривать возможность интеграции частей совпадающих кодов вершин в сопоставляемых инциденторах $F(d_i^k)$ и $F(d_j^k)$. При интеграции допускается также частичное совпадение кодов атрибутов соответствующих дуг.

2. Подстановка схождения $Z_{GH} = \{(e_i, e_j)\}$ графов G и H состоит из пар (e_i, e_j) с вершинами $e_i \in E_d^0(G)$ и $e_j \in E_d^0(H)$, выбранными из пар групп $(E_d^0(G), E_d^0(H))$ с кодами $d_i^0 = d_j^0 = d$. Таким образом, область поиска вариантов пар (e_i, e_j) для включения в Z_{GH} ограничивается величиной $\sum_d (|E_d^0(G)| \times |E_d^0(H)|)$. При этом для каждого d число выбираемых пар (e_i, e_j) не превышает значение $\min(|E_d^0(G)|, |E_d^0(H)|)$.

3. Выбор пар (e_i, e_j) для включения в Z_{GH} сводится к поиску наилучшего соответствия кодов в инциденторах $F(d_i^k)$ и $F(d_j^k)$ среди пар групп в множестве $\{(E_d^k(G), E_d^k(H))\}$. Степень соответствия кодов в инциденторах $F(d_i^k)$ и $F(d_j^k)$, $d_i^k = d_j^k = d$ оценивается величиной $r_{ij}(d)$:

$$r_{ij}(d) = (r_d + r_w / g_w + r_b / g_b + r_h / g_h) / g_{ij}. \quad (1)$$

Здесь r_d – число пар взаимно однозначного соответствия вершин с равными кодами d , взятых из i -го и j -го инцидентора;

r_w, r_b, r_h – числа пар кодов по соответствующим атрибутам дуг W, b, h , которые оказались равными для выделенных в числе r_d пар вершин;

g_w, g_b, g_h – минимальные количества кодов атрибутов дуг среди пар совокупностей (W) , (W, b) или (W, b, h) , приписанных к вершинам, которые образовали пары с равными кодами W, b, h в количестве r_w, r_b, r_h соответственно;

g_{ij} – минимальное число вершин в одном из сравниваемых инциденторов $F(d_i^k)$ и $F(d_j^k)$.

Например, для инциденторов $(2(1,3), 4(8), 6(2, \emptyset, 3))$ и $(2(1,4), 4(8,14), 6(2), 6(2, \emptyset, 3))$, согласно (1), $r_{ij}(d) = (3 + 1/2 + 1/2 + 1/3 + 1/3) / 3 = 5/3$. Заметим, что вершина $4(8)$ в паре с вершиной $4(8,14)$ воспринимается как $4(8, \emptyset)$.

В множестве пар $\{(E_d^k(G), E_d^k(H))\}$, выбор пары для продолжения дифференциации вершин осуществляется по наибольшему значению $r_{ij}^*(d)$:

$$r_{ij}^*(d) = \max_d \{ \max_{i,j} \{ r_{ij}(d) \} \}. \quad (2)$$

Поиск подстановки схождения Z_{GH}^* с наибольшим числом совпадающих дуг в общей части осуществляется на дереве траекторий процесса дифференциации вершин [14].

Изложенный выше подход позволяет разработать единый алгоритм решения задачи выделения общей части, инвариантный относительно специфики ГМ₁ и ГМ₂ и состава атрибутов. Общей частью Q принимаются совпадающие в графах G_1 и G_2 частичные подграфы с наибольшим числом дуг.

Алгоритм выделения общей части

Исходными данными для алгоритма являются описания ГМ₁ и ГМ₂ в виде ориентированных графов G_1 и G_2 . Графы представлены списками инциденторов вершин $\{e_i(F(e_i))\}$, множествами записей кодов атрибутов вершин $\{e_i(V_i, x_i, y_i, \dots)\}$ и дуг $\{(e_j, e_i)(W_{ji}, b_i, h_j)\}$.

Алгоритм включает следующие операции:

1. Формирование графов G и H . Для этого у всех пар вершин $(e_i, e_j) \in (E_{G_1} \times E_{G_2})$ сопоставляются записи в множествах $\{e_i(V_i, x_i, y_i, \dots)\}_{G_1}$ и $\{e_j(V_j, x_j, y_j, \dots)\}_{G_2}$. Если совпадающих записей не найдено, то вершины e_i и e_j исключаются из графов G_1 и G_2 . Сохранившиеся подмножества вершин в графах G_1 и G_2 и дуг между этими вершинами представляют графы G и H .

2. Определение векторов $\mathbf{D}_G^k(F)$ и $\mathbf{D}_H^k(F)$ начальных состояний моделей структур графов G и H . Графы представлены списками инциденторов вершин $\{e_i(F(e_i))\}_G$ и $\{e_j(F(e_j))\}_H$, множествами записей кодов атрибутов вершин $\{e_i(V_i, x_i, y_i, \dots)\}_G$ и $\{e_j(V_j, x_j, y_j, \dots)\}_H$ и дуг $\{(e_q, e_i)(W_{qi}, b_i, h_q)\}_G$ и $\{(e_r, e_j)(W_{rj}, b_j, h_r)\}_H$. Каждой записи $\{e_i(V_i, x_i, y_i, \dots)\}_G$ для $e_i \in E_G$ и записи $\{e_j(V_j, x_j, y_j, \dots)\}_H$ для $e_j \in E_H$ назначаются коды d_i^0, d_j^0 – интегральные описатели кодов атрибутов данных вершин. Далее по спискам $\{e_i(F(e_i))\}_G$ и $\{e_j(F(e_j))\}_H$ формируются списки $\{d_i^0(F(d_i^0))\}_G$ и $\{d_j^0(F(d_j^0))\}_H$ векторов $\mathbf{D}_G^0(F)$ и $\mathbf{D}_H^0(F)$. При этом каждой вершине $e_q \in F(d_i^0)$ и $e_r \in F(d_j^0)$ приписываются записи из множеств $\{(W_{qi}, b_i, h_q)\}_G$ и $\{(W_{rj}, b_j, h_r)\}_H$ для соответствующих дуг (e_q, e_i) и (e_r, e_j) .

3. Выполнение шага взаимозависимой дифференциации вершин путём преобразований $\mathbf{D}_G^0(F) \Rightarrow \mathbf{D}_G^1(F)$ и $\mathbf{D}_H^0(F) \Rightarrow \mathbf{D}_H^1(F)$. В векторах $\mathbf{D}_G^0(F)$ и $\mathbf{D}_H^0(F)$ выделяются пары вершин (e_i, e_j) с уникальными и совпадающими записями $d_i^0(F(d_i^0)) = d_j^0(F(d_j^0))$. Для таких вершин принимается $d_i^1 = d_i^0$ и $d_j^1 = d_j^0$. Далее выделяются пары вершин с уникальными записями, у которых

$d_i^0 = d_j^0$, а инциденторы $F(d_i^0)$ и $F(d_j^0)$ не совпадают частично или полностью. Для этих вершин также принимается $d_i^1 = d_i^0$ и $d_j^1 = d_j^0$.

Вершины, не получившие коды d_i^1, d_j^1 , разбиваются на пары базовых групп $(E_d^0(G), E_d^0(H))$. Среди них выбираются пары, которые не содержат переходы на альтернативные траектории дифференциации вершин. Этим свойством обладают пары, в которых одна из групп содержит одну запись, совпадающую частично или полностью только с одной записью в другой группе. Соответствующие вершины e_i и e_j получают коды $d_i^1 = d_j^1 = d$.

Далее выделяются пары групп, у которых величины $r_{ij}(d)$, вычисленные по (1), однозначно устанавливают соответствие в парах вершин $\{(e_i, e_j)\}$. При этом одна пара (e_i, e_j) получает коды $d_i^1 = d_j^1 = d$, а остальные получают коды $d_i^1 = d_j^1$, равные одному из незанятых кодов d_i^1 в последовательности $(1, 2, \dots, |E_G|)$.

Среди оставшихся пар групп, согласно (2), выбирается пара $(E_d^0(G), E_d^0(H))$ с наибольшим значением $r_{ij}^*(d)$. Вершины (e_i, e_j) , соответствующие величине $r_{ij}^*(d)$, получают коды $d_i^1 = d_j^1 = d$. Вершины (e_i, e_j) с $r_{ij}(d) \leq r_{ij}^*(d)$ получают маскирующие коды $d_i^1 = d_j^1 = d^*$ и включаются в множество E_*^1 в качестве точек запуска альтернативных траекторий.

4. Выполнение шага дифференциации $\mathbf{D}_G^1(F) \Rightarrow \mathbf{D}_G^2(F)$, $\mathbf{D}_H^1(F) \Rightarrow \mathbf{D}_H^2(F)$. Этот и последующие шаги $\mathbf{D}_G^k(F) \Rightarrow \mathbf{D}_G^{k+1}(F)$, $\mathbf{D}_H^k(F) \Rightarrow \mathbf{D}_H^{k+1}(F)$ выполняются операцией 3 до полной дифференциации вершин графа G . При этом на каждом k -м шаге уникальные коды d_i^k и d_j^k сохраняются, а коды d_i^{k+1} и d_j^{k+1} назначаются из незанятых значений множества $(1, 2, \dots, |E_G|)$. Для состояния полной дифференциации $\mathbf{D}_G(F)$ и $\mathbf{D}_H(F)$ определяется подстановка сходства Z_{GH} и соответствующая общая часть Q с числом дуг m_Q .

5. Поиск подстановки сходства Z_{GH}^* с наибольшим числом дуг m_Q^* в общей части Q . Если совокупность $\{E_*^k\}$ содержит множество точек запуска $E_*^k \neq \emptyset$, то дифференциация выполняется также и по альтернативным траекториям. При этом дерево траекторий может просматриваться полностью либо частично с применением схемы поиска по методу ветвей и границ [14, 15].

Работу алгоритма покажем на примере выделения общей части в графах G и H , которые визуальным образом представлены на рис. 2. Здесь же приведены списки кодов атрибутов вершин и дуг.

Графы на рис. 2 содержат четыре вида вершин, $V_i, V_j = (1, 2, 3, 4)$, и три вида дуг, $W_{qi}, W_{rj} = (1, 2, 3)$. В списках кодов атрибутов вершин указаны их интегральные описатели d_i^0, d_j^0 . Исходя из определения графов G и H , для каждого d_i^0 должен существовать один или несколько $d_j^0 = d_i^0$. Аналогично для каждого d_j^0 существует один или несколько $d_i^0 = d_j^0$. Из этого, с учётом $|E_G| \leq |E_H|$, следует, что общая часть $Q(G, H)$ может включать все вершины графа G или их часть.

Результаты работы алгоритма выделения общей части в графах G и H (см. рис. 2) приведены на рис. 3. Структуры графов представлены списками инциденторов $\{e_i(F(e_i))\}$ и $\{e_j(F(e_j))\}$, в которых инциденторы включают только входные вершины. Если вершины e_i или e_j не имеют входов, то соответствующие инциденторы $F(e_i) = F(e_j) = \emptyset$. В графе G – это вершины e_7, e_{13} , в графе H – вершины e_1, e_5 .

Начальные состояния моделей структур $\{d_i^0(F(d_i^0))\}, \{d_j^0(F(d_j^0))\}$ получены в результате замены номеров вершин e_i, e_j в списках инциденторов на соответствующие интегральные описатели d_i^0, d_j^0 . При этом к описателям $d_q^0 \in F(d_i^0)$ и $d_r^0 \in F(d_j^0)$ приписываются коды атрибутов дуг (W_{qi}, b_i, h_q) и (W_{rj}, b_j, h_r) .

Результаты выполнения операции 3 алгоритма по назначению d_i^1 и d_j^1 при переходе $\mathbf{D}^0(F) \Rightarrow \mathbf{D}^1(F)$ отражены на рис. 3 отдельными колонками по каждому из четырёх признаков. По признаку уникальных совпадающих инциденторов назначены коды (5, 8, 6, 9, 3, 4, 11). По признаку совпадения уникальных кодов вершин с разными и частично или полностью совпадающими инциденторами назначены коды (7, 10). По следующему признаку сопоставляются группы с числом вершин $|E_1^0(G)|=1$ и $|E_1^0(H)|=3$, которые не содержат переходы на альтернативные траектории, так как группа $E_1^0(G)$ включает одну вершину. Пара вершин (e_2, e_6) из этих групп получает код $d_2^1 = d_6^1 = 1$.

По отдельному четвёртому признаку сопоставляется другая пара групп $(E_2^0(G), E_2^0(H))$, которая также оказалась без альтернативных траекторий. Здесь для одной пары вершин (e_8, e_4) с наибольшим значением $r_{8,4}(2) = (3+3/1)/3 = 2$ сохранён код $d_8^1 = d_4^1 = 2$. Пара вершин (e_6, e_{10}) с оценкой $r_{6,10}(2) = (2+2/1)/2 = 2$ получила минимальный незанятый код $d_6^1 = d_{10}^1 = 12$. Оставшаяся пара вершин (e_{10}, e_{12}) с оценкой $r_{10,12}(2) = (1+1/1)/2 = 1$ получает код $d_{10}^1 = d_{12}^1 = 13$.

Пример на рис. 3 не содержит пар групп, порождающих альтернативные траектории.

Рис. 2. Примеры графов G и H и списков кодов атрибутов вершин и дуг

$G: e_i(F(e_i))$	$d_i^0(F(d_i^0))$			$d_i^1(F(d_i^1))$	$Z_{GH}^*: (e_i, e_j) \rightarrow d_i$
1(8)	7(2(1))		7	(2(1))	(1,2) → 7
2(1)	1(7(1))		1	(7(1))	(2,6) → 1
3(2)	5(1(1, ∅, 1))	5		(1(1, ∅, 1))	(3,9) → 5
4(6)	8(2(1))	8		(12(1))	(4,11) → 8
5(2)	6(1(1, ∅, 2))	6		(1(1, ∅, 1))	(5,7) → 6
6(3,5)	2(5(1), 6(1))		12	(5(1), 6(1))	(6,10) → 12
7(∅)	9(∅)	9		(∅)	(7,1) → 9
8(5,7,13)	2(6(1), 9(1), 11(3))		2	(6(1), 9(1), 11(3))	(8,4) → 2
9(8)	3(2(1))	3		(2(1))	(9,8) → 3
10(6,9)	2(2(2), 3(1))		13	(3(1), 12(2))	(10,12) → 13
11(4,10)	10(8(3), 2(3))		10	(13(3), 8(3))	(11,15) → 10
12(10)	4(2(1))	4		(13(1))	(12,13) → 4
13(∅)	11(∅)	11		(∅)	(13,5) → 11

$H: e_j(F(e_j))$	$d_j^0(F(d_j^0))$			$d_j^1(F(d_j^1))$	$Q: d_i(F(d_i))$
1(∅)	9(∅)	9		(∅)	1(7)
2(3)	7(1(1, ∅, 2))		7	(*)	2(6,9,11)
3(1,4)	1(9(1), 2(2))				3(12)
4(1,5,7)	2(9(1), 11(3), 6(1))			(6(1), 9(1), 11(3))	4(13)
5(∅)	11(∅)	11		(∅)	5(1)
6(2)	1(7(1))		1	(7(1))	6(1)
7(7)	6(1(1, ∅, 2))	6		(1(1, ∅, 2))	7(*)
8(4)	3(2(1))	3		(2(1))	8(12)
9(6)	5(1(1, ∅, 1))	5		(1(1, ∅, 1))	9(0)
10(7,9)	2(6(1), 5(1))		12	(5(1), 6(1))	10(13)
11(10)	8(2(1))	8		(12(1))	11(0)
12(8,11)	2(3(1), 8(1))		13	(3(1), 8(1))	12(5,6)
13(12)	4(2(1))	4		(13(1))	13(3)
14(9)	1(5(1))				
15(12,14)	10(2(3), 1(3))		10	(13(3), (*))	

Рис. 3. Выделение общей части в графах G и H , приведенных на рис. 2

Поэтому состояния $D_G^1(F)$ и $D_H^1(F)$ соответствуют подстановке схождения Z_{GH}^* (на рис. 2 приведена справа вверху). Список инциденторов $\{d_i(F(d_i))\}$ для общей части $Q(G, H)$ с $m_Q = 13$ приведен справа внизу. Вершины общей части в графах G и H в изображениях вершин на рис. 2 указаны кодами d_i , а дуги, вошедшие в общую часть, отмечены жирным.

Заключение

В статье предложен метод решения задачи выделения общей части в версиях программной системы, представленных графовыми моделями (ГМ). Разработаны правила преобразования ГМ, представленных описаниями структур и списками атрибутов вершин и дуг, в стандартную форму в виде ориентированных графов, пригодную для применения взаимозависимой дифференциации вершин. Это позволило разработать алгоритм выделения общей части в ориентированных графах, который является инвариантным по отношению к особенностям описаний структур и набору атрибутов, используемых в сравниваемых ГМ.

Полученные результаты имеют важное значение для дальнейшего развития и расширения функций системы управления версиями. К ним можно отнести функции контроля исходного кода, оптимизации его структуры, визуального анализа фрагментов, иерархической структуризации исходного кода и получения временных оценок функционирования, разработки планов тестирования и ряд других функций по сопровождению разработки программ.

Литература

1. Власов А.И. Системный подход к проектированию при каскадной и итеративной модели жизненного цикла / А.И. Власов, А.А. Карпунин, Ю.М. Ганев // Тр. междунар. симпозиума «Надежность и качество». – 2015. – Т. 1. – С. 96–99.
2. Брагина Т.И. Сравнительный анализ итеративных моделей разработки программного обеспечения / Т.И. Брагина, Г.В. Табунчик // Радиоэлектроника, информатика, управления. – 2010. – № 2(23). – С. 130–138.
3. Бахтизин В.В. Модель жизненного цикла программных средств при разработке на основе гибких методов / В.В. Бахтизин, С.Н. Неборский // Доклады БГУИР. – 2008. – № 8(38). – С. 65–69.
4. Сиротенко Ф.Ф. Принципы работы и возможности системы управления версиями // Научные технологии в космических исследованиях Земли. – 2010. – № 1. – С. 15–17.
5. Шурыгин В.Н. Системы контроля версий / В.Н. Шурыгин, Д.А. Сиваченко // Вестник МГУП. – 2015. – № 5. – С. 103–104.
6. Chacon S. Pro Git (2nd ed.) / S. Chacon, B. Straub. – Berkeley, CA, USA: Apress, 2014. – 456 p.
7. O'Sullivan B. Mercurial: The Definitive Guide. – O'Reilly Media, 2009. – 284 p.
8. Gold R. Control flow graphs and code coverage // Applied Mathematics and Computer Science. – 2010. – No. 4. – P. 739–749.
9. Feist M.D. Visualizing Project Evolution through Abstract Syntax Tree Analysis / M.D. Feist, E.A. Santos, I. Watts, A. Hindle // 2016 IEEE Working Conference on Software Visualization. – 2016. – P. 11–20.
10. Касьянов В.Н. Графы в программировании: обработка, визуализация и применение / В.Н. Касьянов, В.А. Евстигнеев. – СПб.: БХВ-Петербург, 2003. – 1104 с.
11. Pogrebnoy A.V. Constructing graph models for software system development and analysis // 2017 J. Phys.: Conf. Ser. – URL: <http://iopscience.iop.org/article/10.1088/1742-6596/803/1/012118/pdf> (дата обращения: 03.10.2018).
12. Дэмер М. Новый функционал информативности для анализа структуры химических графов / М. Дэмер, Ф. Эммерт-Штрайб, Ю.Р. Цой, К. Вармуза // Изв. Том. политехн. ун-та. – 2010. – Т. 316, № 5. – С. 5–11.
13. Dehmer M. Quantum Frontiers of Atoms and Molecules / M. Dehmer, F. Emmert-Streib, R.Y. Tsoy, K. Varmuza. – New York: Nova Publishing, 2011. – 673 p.
14. Погребной А.В. Метод определения схождения структур графов на основе выделения частичного изоморфизма в задачах геоинформатики // Изв. Том. политехн. ун-та. – 2015. – Т. 326, № 11. – С. 56–66.
15. Land A.H. An Automatic Method of Solving Discrete Programming / A.H. Land, A.G. Doig // Problems Econometrica. – 1960. – Vol. 28, No. 3. – P. 497–520.

Погребной Андрей Владимирович

Аспирант инженерной школы информационных технологий и робототехники (ИШИТР)
Томского политехнического университета (ТПУ)
Ленина пр-т, д. 30, г. Томск, Россия, 634050
Тел.: +7-952-808-36-82
Эл. почта: avpogrebnoy@gmail.com

Pogrebnoy A.V.

Finding common parts in software source code versions represented by graph models

Solving the problem of the objects structural similarity analysis based on graph models (GM) is one of the actual research directions of the applied graph theory. In the paper, program source code versions are used as objects for similarity analysis. Existing version control systems (VCS) work directly with the text of the program source code. Giving VCS an ability to work with GM, adequately representing program versions source code, opens new ways to improve traditional functions of the VCS and expand program development tools. It is proposed to base program versions similarity estimation on allocating biggest common part in the program source code. The aim of the study is to develop the method of the common part allocation if the program versions, represented as a GM. Research is based on the application of the graph vertices differentiation theory, where graph structure model is represented as a network of the state machines. Important method's advantage is an ability to easily adapt to program source code features. Method includes three stages – forming and encoding list of attributes of GM vertices and edges; transforming GM into standard form as a directed graph; allocating common part in two directed graphs, representing compared program versions. Algorithm of allocating common part is invariant relatively to GM description features and set of used attributes. Example of solving the problem of the common part allocation for two compared GM fragments is provided.

Keywords: software graph model, version control system, common part of two graphs, graph similarity substitution, graph nodes differentiation.

doi: 10.21293/1818-0442-2018-21-3-49-56

References

1. Vlasov A.I., Karpunin A.A., Ganey Yu.M. Sistemniy podkhod k proektirovaniyu pri kaskadnoy i iterativnoy modeli zhiznennogo tsikla. *Trudi Mezhdunarodnogo simpoziuma «Nadezhnost i kachestvo»*, 2015, vol. 1, pp. 96–99 (In Russ.).
2. Bragina T.I., Tabunshchik G.V. Sravnitelniy analiz iterativnih modeley razrabotki programmnogo obespecheniya. *Radioelektronika, informatika, upravlenie*, 2010, no. 2 (23), pp. 130–138 (In Russ.).
3. Bakhtizin V.V., Neborskiy S.N. Model zhiznennogo tsikla programmnikh sredstv pri razrabotke na osnove gibkikh metodov. *Doklady BGyUR*, 2008, no. 8 (38), pp. 65–69 (In Russ.).
4. Sirotenko F.F. Principy raboty i vozmozhnosti sistemi upravleniya versiyami. *Naukoemkie tekhnologii v kosmicheskikh issledovaniyakh Zemli*, 2010, no. 1, pp. 15–17 (In Russ.).
5. Shurygin V.N., Sivachenko D.A. Sistemi kontrolya versiy. *Vestnik MGUP*, 2015, no. 5, pp. 103–104 (In Russ.).
6. Scott Chacon and Ben Straub. *Pro Git* (2nd ed.). Berkely, CA, USA: Apress, 2014. 456 p.
7. Bryan O'Sullivan. *Mercurial: The Definitive Guide*. O'Reilly Media. 2009, 284 p.
8. Gold, Robert. Control flow graphs and code coverage. *Applied Mathematics and Computer Science*. 2010. No. 4, pp. 739–749.
9. Feist M.D., Santos E.A., Watts I., Hindle A. Visualizing Project Evolution through Abstract Syntax Tree Analysis. *2016 IEEE Working Conference on Software Visualization*, 2016, pp. 11–20.
10. Kasyanov V.N., Evstigneev V.A. *Grafi v programmirovanii: obrabotka, vizualizatsiya i primeneniye*. SPb.: BKhV-Peterburg, 2003, 1104 p. (In Russ.).
11. Pogrebnoy A.V. Constructing graph models for software system development and analysis. *2017 J. Phys.: Conf. Ser.* Available at: <http://iopscience.iop.org/article/10.1088/1742-6596/803/1/012118/pdf> (accessed: October 3, 2018).
12. Demer M., Emmert-Shtraib F., Tsoy R.Yu., Varmuza K. Noviy funktsional informativnosti dlya analiza struktury khimicheskikh grafov. *Bulletin of Tomsk Polytechnic University*, 2010, vol. 316, no. 5, pp. 5–11 (In Russ.).
13. Dehmer M., Emmert-Streib F., Tsoy R.Y., Varmuza K. *Quantum Frontiers of Atoms and Molecules*. New York, Nova Publishing, 2011, 673 p.
14. Pogrebnoy An.V. Metod opredeleniya skhodstva struktur grafov na osnove vydeleniya chastichnogo izomorfisma v zadachakh geoinformatiki. *Bulletin of Tomsk Polytechnic University*, 2015, vol. 326, no. 11, pp. 56–66 (In Russ.).
15. Land A.H., Doig A.G. An Automatic Method of Solving Discrete Programming. *Problems Econometrica*, Vol. 28, no. 3, 1960, pp. 497–520.

Andrey V. Pogrebnoy

PhD student, School of Computer Science & Robotics,
Tomsk Polytechnic University
30, Lenin pr., Tomsk, Russia, 634050
Phone: +7-952-808-36-82
Email: avpogrebnoy@gmail.com