ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ НАУК

МАТЕМАТИЧЕСКИЕ СТРУКТУРЫ И МОДЕЛИРОВАНИЕ

Выпуск 21

Омск 2010

Журнал «**Математические структуры и моделирование**». – Омск: «Омское книжное издательство», 2010. – Вып. 21. – 183 с.

ISBN 978–5–85540–609–2

Художественное оформление
В.В. Коробицын

Адрес научной редакции

Россия, 644053, Омск-53, ул. Грозненская, 11
Омский государственный университет
факультет компьютерных наук

E-mail: guts@omsu.ru
msm@cmm.univer.omsk.su

ISBN 978–5–85540–609–2

МАТЕМАТИЧЕСКИЕ СТРУКТУРЫ и МОДЕЛИРОВАНИЕ

В журнале публикуются статьи, в которых излагаются результаты исследований по фундаментальной и прикладной математике, теоретической физике и размышления, касающиеся окружающей нас природы и общества.

Публикуются также статьи по информационным технологиям, компьютерным наукам, защите информации, философии и истории математики.

Объекты исследования должны быть представлены в форме некоторых математических структур и моделей.

Журнал является реферируемым. Рефераты статей публикуются в «Реферативном журнале» и в журналах «Zentralblat für Mathematik» (Германия) и «Mathematical Reviews» (США).

Электронная версия журнала представлена в сети Интернет по адресу:

`http://cmm.univer.omsk.su`

Журнал издается на коммерческие средства факультета компьютерных наук Омского государственного университета.

Наш E-mail:

`msm@cmm.univer.omsk.su`

Подробную информацию можно найти на Web-сервере:

`http://cmm.univer.omsk.su`

СОДЕРЖАНИЕ

Фундаментальная и прикладная математика

Компьютерные науки

Продолжение на следующей странице

Информационная безопасность

Информационные технологии в образовании

Математические структуры и моделирование
2010, вып. 21, с. 5–10

УДК 519.17

МЕТРИЧЕСКАЯ СВЯЗНОСТЬ ВЕРШИН ГРАФА

С.В. Белим, В.Ю. Бардычев

В данной работе проводится исследование одного из возможных вариантов функции, определяющей меру связности вершин взвешенного графа. Исследования проводятся на основе компьютерного эксперимента.

Введение

При исследовании графов важную роль играет понятие связности, понимаемое как существование пути из одной вершины графа в другую. Однако в ряде прикладных задач важным становится не только сама достижимость из одной вершины другой вершины, но и длина соответствующего пути.

Будем считать, что дуги определяют некоторые взаимодействия или связи между вершинами. Поставим себе задачу нахождения не только непосредственных связей между вершинами, но и связи через посредников, то есть более длинные пути. В качестве приложений данной проблемы можно привести квантовые системы, в которых роль множества вершин играют состояния системы, а множества дуг – переходы между состояниями. При этом возможны переходы как напрямую между состояниями, так и через цепочку промежуточных состояний. Другим примером является связь между научными работами по одной тематике. Подавляющее количество научных работ основывается на результатах, полученных в более ранних работах. Причем авторы отдельно взятой работы могут основываться на результатах других авторов непосредственно, а могут через цепочку промежуточных результатов третьих авторов.

1. Постановка задачи

Пусть граф определен множеством вершин V и множеством дуг D. Отношение инцидентности будем задавать с помощью матрицы смешения E (mixing matrix) [1]. А именно, если существует дуга из вершины v_i в вершину v_j, то соответствующий элемент матрицы смешения $E_{ij} = 1$, в противном случае $E_{ij} = 0$. Также будем считать, что в графе нет петель, и полагать $E_{ii} = 0$.

Попытаемся получить числовую характеристику, показывающую степень связности двух вершин. Причем это не просто количество вершин или дуг, удаление которых к тому, что заданные вершины попадают в разные компоненты

Омский государственный университет им. Ф.М. Достоевского.
E-mail: sbelim@mail.ru

связности. Наша оценка должна учитывать длины путей, соединяющих эти вершины. Соответствующую числовую характеристику будем называть метрической связностью.

Пусть между вершинами v_i и v_j некоторого графа существует несколько путей длины k. Обозначим количество таких путей через $p_k(v_i, v_j)$. В общем случае будем считать, что пути разной длины дают разный вклад в метрическую связность двух вершин. Введем функцию $g(k)$, которую в дальнейшем будем называть весовой функцией пути. Более точно $g(k)$ показывает вклад в связь вершин пути длиной k. Тогда метрическая связность двух вершин может быть определена следующим образом:

$$w(v_i, v_j) = \sum_{k=1}^{\infty} g(k) p_k(v_i, v_j).$$

Рассматривая граф как представление влияния элементов множества V друг на друга можно сказать, что наличие дуги между вершинами показывает прямое влияние, путь длины 2 – влияние через одного посредника, длины 3 – через двух посредников и так далее, длины n — через $n-1$ посредника.

Определим матрицу метрической связности вершин M, показывающую взаимное влияние всех вершин графа друг на друга:

$$M_{ij} = w(v_i, v_j).$$

Как показано в работе [2], матрица метрической связности может быть вычислена на основе матрицы смешения:

$$M = \sum_{k=1}^{\infty} g(k) E^k.$$

Матрица метрической связности играет роль, аналогичную матрице достижимости [2], служащей для определения связности графа.

При определении метрической связности важным является вопрос выбора весовой функции пути $g(k)$. Рассмотрим свойства, которым должна удовлетворять эта функция:

1. $g(1) = 1$, то есть наличие дуги между двумя вершинами должно давать единичный вклад в метрическую связность.

2. $g(k) > g(k+1)$, для всех $k \in [1, \infty)$. То есть функция $g(k)$ должна быть убывающей.

3. $g(k) > 0$, для всех $k \in [1, \infty)$. Наличие путей должно увеличивать метрическую связность.

Рассмотрим два возможных варианта выбора функции $g(k)$.

1. $g(k) = 1/\alpha^{k-1}$, где $0 < \alpha < 1$. Функция данного вида была рассмотрена в [1] для выявления связных структур методом построения иерархического дерева. В этом случае метрическая связность двух вершин вычисляется по формуле

$$w(v_i, v_j) = \sum_{k=1}^{\infty} \alpha^{k-1} p_k(v_i, v_j).$$

Матрица метрической связности имеет вид:

$$M = \sum_{k=1}^{\infty} \alpha^{k-1} E^k = \alpha^{-1}((I - \alpha E)^{-1} - I).$$

Здесь I – единичная матрица. Коэффициент α имеет двоякий смысл. С одной стороны, он показывает, во сколько раз уменьшается относительное влияние пути при увеличении его длины на единицу. С другой стороны, рассмотрим ситуацию, при которой из вершины v_i в вершину v_j ведет несколько путей одинаковой длины. В частности, если имеется $p_2(v_i, v_j) = 1/\alpha$ путей длины 2, то они дают такой же вклад в метрическую связность, как и дуга между вершинами. При наличии $p_3(v_i, v_j) = 1/\alpha$ путей длины 3 их вклад будет эквивалентен одному пути длиной 2. Если же имеется $p_3(v_i, v_j) = 1/^2$ путей длины 3, то их вклад эквивалентен одной дуге. В общем случае при наличии $p_n(v_i, v_j) = 1/\alpha^{n-k}$ путей длины n их вклад будет эквивалентен одному пути длины k. Эти соображения позволяют выбирать коэффициент α из практических соображений при постановке задачи.

2. $g(k) = 1/k$. Метрическая связность двух вершин будет вычисляться по формуле

$$w(v_i, v_j) = \sum_{k=1}^{\infty} k^{-1} p_k(v_i, v_j).$$

Соответственно матрица сил связности:

$$M = \sum_{k=1}^{\infty} k^{-1} E^k = -\ln(I - E).$$

В данном случае длинные пути дают значительно больший вклад в метрическую связность, чем в предыдущем случае. В частности, необходимо $p_n(v_i, v_j) = n$ путей длины n, чтобы получить вклад, эквивалентный дуге. И, соответственно, $p_n(v_i, v_j) = n/k$ путей длины n, чтобы получить вклад, эквивалентный пути длиной k.

Нахождение метрической связности вершин может применяться для решения двух основных задач:

Частная задача. Нахождение вершин графа, наиболее сильно связанных с заданной вершиной. Более строго формулировка задачи сводится к поиску вершин v', метрическая связаность которых с данной вершиной v лежит в некотором интервале In:

$$w(v, v') \in In.$$

При данном подходе существенным является выбор интервала In. Проанализируем влияние выбора интервала на смысл решаемой задачи для весовой функции $g(k) = 1/\alpha^{k-1}$ $(0 < \alpha < 1)$.

Рассмотрим несколько вариантов интервалов In:
$In = [1, \infty)$ – позволяет выявить все вершины, связанные с данной вершиной дугами, либо эквивалентные случаи;

$In = (1, \infty)$ – позволяет выявить вершины, которые связаны с данной дугой и еще, как минимум, одним путем;
$In = [1 + \alpha, \infty)$ – вершины, связанные с v дугой и, как минимум, путем с длиной 2;
$In = [1 + \alpha^n, \infty)$ – вершины, связанные с v дугой и, как минимум, путем с длиной не более $n + 1$;
$In = [\alpha, 1)$ – вершины, не связанные дугой с v, но связанные путем длины 2;
$In = [\alpha^2, \alpha)$ – вершины, не связанные дугой с v, но связанные путем длины 3;
$In = [\alpha^n, \alpha^{n-k})$ – вершины, не связанные дугой с v, но связанные путем длины не менее k и не более $n + 1$;

В этом случае k путей длины k эквивалентны дуге.

Общая задача. Разбиение графа на связные структуры, то есть поиск подграфов с определенной метрической связностью вершин. Как и для частной задачи, рассмотрим случаи различных весовых функций.

Необходимо ввести некоторый интервал значений метрической связности In для вершин, входящих в одну связную структуру. Более строго, связной структурой будем называть подмножество вершин $V' \subseteq V$, для каждой пары вершин из которого

$$w(v_i, v_j) \in In, \quad v_i, v_j \in V'.$$

Рассмотрим несколько вариантов интервалов In для весовой функции $g(k) = 1/\alpha^{k-1}$ $(0 < \alpha < 1)$:
$In = [1, \infty)$ – позволяет выявить связные структуры, все вершины которых попарно связаны друг с другом дугой, либо эквивалентные случаи;
$In = (1, \infty)$ – позволяет выявить связные структуры, все вершины которых попарно связаны друг с другом дугой и еще, как минимум, одним путем;
$In = [1+\alpha, \infty)$ – позволяет выявить связные структуры, все вершины которых попарно связаны друг с другом дугой и, как минимум, путем с длиной 2;
$In = [1+\alpha^n, \infty)$ – позволяет выявить связные структуры, все вершины которых попарно связаны друг с другом дугой и, как минимум, путем с длиной не более $n + 1$;
$In = [\alpha, 1)$ – позволяет выявить связные структуры, все вершины которых попарно не связаны друг с другом дугой, но связанные путем длины 2;
$In = [\alpha^2, \alpha)$ – позволяет выявить связные структуры, все вершины которых попарно не связаны друг с другом дугой, но связанные путем длины 3;
$In = [\alpha^n, \alpha^{n-k})$ – позволяет выявить связные структуры, все вершины которых попарно не связаны друг с другом, не связаны дугой с v, но связанные путем длины не менее k и не более $n + 1$;

2. Компьютерный эксперимент

Компьютерный эксперимент ставился с целью сравнения результатов, получаемых при различных весовых функциях $g(k)$. Исследовались ориентированные графы с количеством вершин равным 100 и некоторым определенным количеством дуг d. Матрица смешения задавалась случайным образом. Во всех экспериментах генерировалось 100 матриц, после чего производилось усреднение.

Матрица связности вычислялась с точностью до третьего слагаемого:

$$M^{(3)} = g(1)E + g(2)E^2 + g(3)E^3.$$

На рисунке 1 по оси абсцисс отложена величина метрической связности, а по оси ординат количество соответствующих связей.

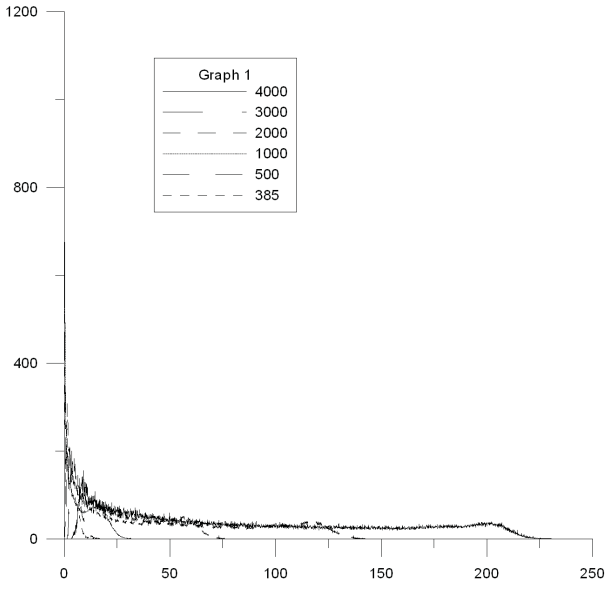


Рис. 1. Решение частной задачи для весовой функции $g(k) = \alpha^{k-1}$ ($\alpha = 0.1$)

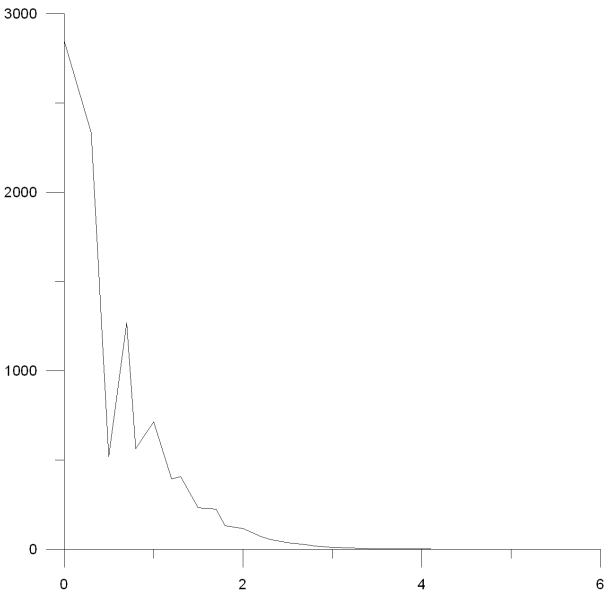


Рис. 2. Решение частной задачи для весовой функции $g(k) = 1/k$, $d = 500$

На рисунках 2 и 3 представлены аналогичные графики для весовой функции $g(k) = 1/k$ и $d = 500$(рис. 2) и $d = 3000$ (рис. 3).

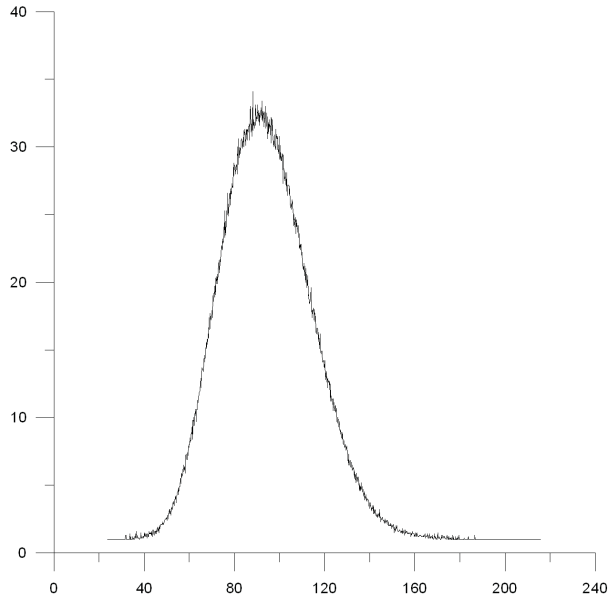


Рис. 3. Решение частной задачи для весовой функции $g(k) = 1/k$, $d = 3000$

Как видно, в случае весовой функции $g(k) = 1/k$ присутствует пик связностей определенной величины. Это обстоятельство делает данный вид весовой функции малопригодным для исследования графов в реальных задачах. Напротив, использование весовой функции $g(k) = \alpha^{k-1}$ приводит к адекватному поведению графиков, а именно: чем больше величина метрической связности, тем меньше соответствующее количество связей. Попытки получения точного результата по формуле

$$M = \alpha^{-1}((I - \alpha E)^{-1} - I)$$

наталкиваются на трудности, связанные с наличием ориентированных циклов, и, как следствие, с многократным учетом одних и тех же путей.

Авторы выражают благодарность Н.Ф. Богаченко за полезные обсуждения постановки задачи и используемой терминологии.

Литература

1. Girvan M., Newman M.E.J. Community structure in social and biological networks // arXiv:cond-mat/0112110v1 (2001).
2. Newman M.E.J. Mixing patterns in networks // Phys. Rev. E. 2003. V. 67. P. 026126-1–026126-13.

Математические структуры и моделирование
2010, вып. 21, с. 11–16

УДК 519.17

ПОИСК СВЯЗАННЫХ СТРУКТУР ВО ВЗВЕШЕННЫХ ГРАФАХ

С.В. Белим, А.В. Сорокин

В данной работе проводится исследование алгоритмов выявления связанных структур во взвешенных графах. На основе компьютерного эксперимента сравниваются алгоритмы полного перебора и «жадный» алгоритм.

Введение

В настоящее время моделирование многих систем опирается на теорию графов, то есть множества вершин и связей между ними. Особенно большое количество моделей разработано для социальных сетей [1–3], компьютерных сетей [4, 5], биологических систем [3] и сетей обслуживания [6]. Ранние исследования были ориентированы на изучение статистических свойств, вытекающих из топологии сети. Такие свойства получили название «эффекты малых сетей», так как исследуемые сети имели малые расстояния между вершинами [7,8], обычно растущие логарифмически с увеличением количества вершин.

Во многих сетях проявляется свойство образования связанных структур (community structure). Это свойство иногда называют кластеризацией, однако мы не будем пользоваться этим термином, так как его принято использовать несколько в ином смысле. Термин «связанные структуры» первоначально появился при исследовании социальных сетей и в дальнейшем получил распространение на другие аналогичные сети. Мы, по аналогии с работами [9–12], под связанными структурами будем понимать подмножество вершин, связанных между собой сильнее, чем с остальными вершинами графа. В данном определении недостаточно четко выглядит понятие «сильнее связаны». В различных сетях для характеристики величины связи вводятся разные функции.

На сегодняшний день разработано несколько подходов к поиску связанных структур:

1. *Полный перебор* состоит в выделении подмножеств вершин и вычислении функции силы связности соответствующей структуры. Этот подход имеет одно преимущество – алгоритм является точным. Однако, как легко понять, сложность алгоритма растет экспоненциально с увеличением числа вершин, в силу чего алгоритм становится не пригодным для достаточно больших графов.

Омский государственный университет им. Ф.М. Достоевского.
E-mail: sbelim@mail.ru

2. *Иерархическая кластеризация* дает более быстрый алгоритм, однако не всегда правильный ответ [11,12]. Метод состоит в том, что сначала вычисляется вес связи для каждой пары вершин. Затем строится многоуровневое дерево, листьями которого являются исходные вершины. На первом шаге построения дерева появляется новая вершина, связанная дугами с двумя наиболее сильно связанными между собой вершинами. Далее в исходном графе образуется стяжка двух вершин, выделенных ранее при построении дерева. Таким образом в графе две вершины заменяются одной, которая наследует все связи с остальными вершинами. То есть если у одной из вершин, попавших в стяжку, была дуга к какой-либо вершине, то она будет и в новом графе. Далее задача образования стяжки из двух вершин решается в новом графе и так далее. Полученное многоуровневое дерево в социологии получило название дендрограммы [12].

3. Генетические алгоритмы [13, 14] были разработаны именно для больших графов и используют метод, аналогичный построению многоуровневого дерева. Вероятность правильного разбиения графа на связанные структуры при этом, как и следовало ожидать, еще ниже, чем во втором случае.

Целью данной статьи является построение и исследование с помощью компьютерного эксперимента «жадного» алгоритма поиска связанных структур.

1. Описание графов

Начнем с представления графов, на которых будут выявляться связанные структуры. Будем считать, что в графе заданы веса как ребер, так и вершин. Такие графы могут быть заданы с помощью матрицы смешения E, предложенной в работе [10]. Элементы матрицы смешения задаются следующим образом. Диагональный элемент E_{ii} показывает вес вершины с номером i (v_i). Элемент E_{ij} ($i \neq j$) показывает величину связи вершины v_i с вершиной v_j. Как показано в работе [10], более удобным является приведенный вид матрицы смешения $e = E/m$, где $m = \sum E_{ij}$. В приведенной матрице смешения элемент e_{ij} показывает долю веса заданного ребра в общем весе графа. В дальнейшем под матрицей смешения будет пониматься именно приведенный вид. Легко увидеть, что $\sum e_{ij} = 1$.

Способы задания матрицы смешения могут быть разными и зависят от алгоритма определения величины связи вершин. Так, в работе [10] в качестве величины связи двух вершин используется вес связывающего их ребра. При этом вес вершины можно трактовать как суммарный вес петель с концами в этой вершине. В работе [15] в качестве величины связи двух вершин используется сумма весов всех путей в графе, ведущих из одной вершины в другую с весовыми коэффициентами, зависящими от длины пути. Влияние способа построения матрицы смешения на выявление связанных структур до сих пор остается не до конца исследованным.

2. Мера связности вершин

Как уже было сказано во введении, для выявления связанных структур необходимо определить некоторую функцию от элементов матрицы смешения, численно определяющую силу связности. Будем считать, что такая функция задана при постановке задачи, и будем обозначать ее $Q(e)$ и называть мерой связности.

В ряде работ было предложено несколько функций меры связности вершин.

1. Метод парных корреляций был предложен в работе [16] и состоит в вычислении коэффициентов Жаккарда и индексов Ранда для пар вершин, взятых из различных подграфов исходного графа.

2. Метод кластеризации, основанный на метрике Донгена [17].

3. Теоретико-информационный подход [18, 19], рассматривающий меру связности как интенсивность обмена информацией. Далее на основе вычисления взаимной энтропии выделяются связанные структуры, внутри которых обмен информацией происходит интенсивнее, чем с остальными вершинами.

4. Метод Ньюмана, исследованный в работах [10–12]. В качестве меры связанности используется величина

$$Q(e) = \sum e_{ii} - \sum a_i b_i,$$

где

$$a_i = \sum e_{ij}, b_i = \sum e_{ji}.$$

Выбор функции меры связанности графа зависит от постановки задачи. В данной работе в качестве функции $Q(e)$ мы будем использовать выражение, предложенное в работах Ньюмана.

3. Задача поиска связанных структур

Определим более строго процедуру образования связанной структуры в графе. Начнем с алгоритма образования стяжек как преобразования графа G в граф $G1$. Выделим в графе G подграф G' и заменим все входящие в него вершины одной вершиной, при этом вершины подграфа $G \setminus G'$ остаются неизменными. Образованная вершина связана дугами с теми вершинами графа $G1$, с которыми были связаны вершины, вошедшие в стяжку. Вес вершины, вошедшей в стяжку, равен сумме весов вершин и дуг, вошедших в стяжку. При этом, если граф не ориентированный, при образовании стяжки каждую дугу заменяем двумя дугами с противоположной ориентацией и одинаковым весом.

Под связанной структурой будем понимать подграф исходного графа, который при образовании из него стяжки максимизирует меру связанности графа. Будем различать две задачи выявления связанных структур — частную и общую.

Частная задача:
Поиск связанной структуры в исходном графе, включающей в себя заданную вершину.

Общая задача:
Выявление всех связанных структур в графе.

В данной работе задачи выявления связанных структур в графах решались с помощью компьютерного эксперимента. Рассматривались взвешенные графы с различным количеством вершин. Для каждого размера графа случайным образом генерировалось по 100 матриц смешения. После чего решалась задача выявления связанных структур.

Достаточно сложным является вопрос: является ли частная задача частью общей задачи? То есть всегда ли связанная структура, образованная при решении частной задачи, сохранится при решении общей задачи. Для проверки этой гипотезы был проведен компьютерный эксперимент. Последовательно для всех вершин графа решалась частная задача. Затем для всего графа решалась общая задача и проверялось, все ли связанные структуры, полученные при решении частных задач, присутствуют в решении общей задачи.

4. «Жадный» алгоритм поиска связанных структур

Несложно заметить, что точный алгоритм, построенный на полном переборе, имеет экспоненциальную сложность. Поэтому возникает задача построения других алгоритмов, дающих точное решение задачи либо решение, близкое к точному.

Рассмотрим следующий «жадный» алгоритм решения частной задачи для вершины v.

1. Ищем вершину v_1, связанную с v дугой, которая при образовании стяжки с v дает наибольшее увеличение меры связности Q.

2. Образуем стяжку из вершин v_1 и v, обозначаем ее через v и переходим к пункту 1.

3. Пункты 2 и 3 повторяем до тех пор, пока существуют вершины, стяжка с которыми увеличивает Q.

Для определения эффективности «жадного» алгоритма был проведен компьютерный эксперимент решения частной задачи поиска связанных структур с помощью «жадного» алгоритма и точного алгоритма (перебора).

«Жадный» алгоритм решения общей задачи поиска связанных структур выглядит следующим образом.

1. Выбираем одну из вершин графа и решаем для нее частную задачу поиска связанных структур.

2. В графе, полученном в результате стяжки связанной структуры, найденной в первом пункте, выбираем вершину, отличную от выбранной ранее, и для нее решаем частную задачу поиска связанных структур.

3. Повторяем пункт 2 до тех пор, пока все вершины не будут определены в связанные структуры (связанные структуры могут содержать и одну вершину).

На рисунке приведены сравнительные результаты компьютерного эксперимента. Тест 1 показывает процент случаев, в которых решение частной задачи присутствует в решении общей задачи для матриц различного размера. Тест 2 демонстрирует процент совпадения решений частной задачи поиска связанных структур «жадным» алгоритмом и полным перебором.

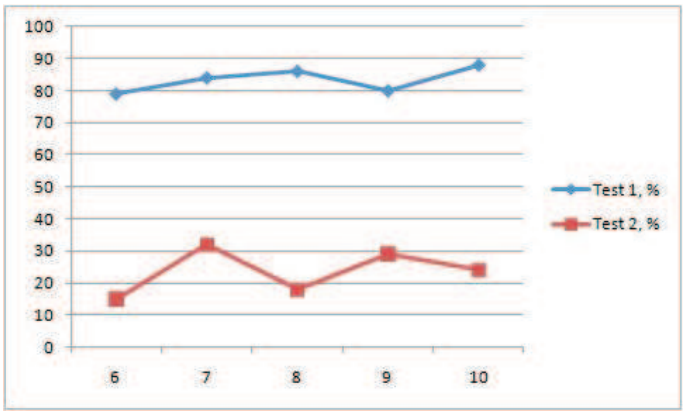


Рис. 1. Сравнение результатов «жадного» алгоритма и точного решения для общей задачи

5. Обсуждение результатов

Результаты компьютерного эксперимента показывают, что объединение в связанные структуры, выгодное одной вершине, не всегда выгодно при полном разбиении графа на связанные структуры (тест 1). Отсюда следует, что решение общей задачи с помощью «жадного» алгоритма не всегда будет совпадать с точным решением.

«Жадный» алгоритм не всегда приводит к точному решению частной задачи выявления связанных структур (тест 2). Однако учитывая, что в случаях несовпадения решений отклонение меры связанности Q «жадного» алгоритма от точного алгоритма составляет не более 5%, можно считать, что «жадный» алгоритм дает хороший результат, не уступающий в точности другим приближенным методам [19].

Литература

1. Wasserman S., Faust K. Social Network Analysis. Cambrige.: Cambrige University Press, 1994.
2. Scott J. Social Network Analysis: A Handbook. London.: Sage Publication, 2000.
3. Watts D. J., Strogatz S. H. Collective dynamics of 'small-world' networks // Nature. 1998. V. 393. P. 440–442.
4. Faloutsos M., Faloutsos P., Faloutsos C. On power-law relationships of the internet topology // Computer Communication Review. 1999. V. 29. P. 251–262.
5. Albert R., Jeong H., Barabasi A.-L. Diameter of the world-wide web // Nature. 1999. V. 401. P. 130–131.
6. Newman M. E. J. The structure of scientific collaboration network // Proc. Natl. Acad. Sci. USA. 2001. V. 98. P. 404–409.
7. Pool I., Kochen M. Contact and influence // Social network. 1978. V. 1. P. 1–48.
8. Milgram S. The small world problem // Psychology Today. 1967. V. 2. P. 60–67.
9. Girvan M., Newman M. E. J. Community structure in social and biological networks // arXiv:cond-mat/0112110v1. (2001)

10. Newman M. E. J., Girvan M. Finding and evaluting community structure in networks // arXiv:cond-mat/0308217v1. (2003)
11. Newman M. E. J. Fast algorithm for detecting community structure in networks // arXiv:cond-mat/0309580v1. (2003)
12. Newman M. E. J. Mixing patterns in networks // Phys. Rev. E. 2003. V. 67. P. 026126-1–026126-13.
13. Berryman M. J., Allison A., Abbott D. Optimizing genetic algorithm strategies for evolving networks // arXiv:cs/0404019v1. (2004)
14. Tasgin M., Herdagdelen A., Bingol H. Community detection in complex network using genetic algorithms // arXiv:0711.0491v1. (2007)
15. Leicht E. A., Holme P., Newman M. E. J. Vertex similarity in networks // arXiv:physics/0510143v1. (2005)
16. Meilia M. Comparing clasterings-an information based distance // Journal of Multivariate Analysis. 2007. V. 98. P. 873–895.
17. Dongen S. V. Performance criteria for graph clastering and Markov cluster experiments. National Research Institute for Mathematics and Computer Science in the Netherlands, 2000.
18. Meilia M. Comparing clusterings: an axiomatic view. // ICML '05: Proceedings of 22nd International Conference on Machine Learning, New York:ACM Press, 2005. P. 577–584.
19. Meilia M. Comparing clasterings // Technical report, University of Washington, 2002.

Математические структуры и моделирование
2010, вып. 21, с. 17–23

УДК 517.91

ЛИНЕЙНОЕ ПРИБЛИЖЕНИЕ ТОЧКИ ПЕРЕСЕЧЕНИЯ КРИВОЙ РЕШЕНИЯ СИСТЕМЫ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ С ПОВЕРХНОСТЬЮ РАЗРЫВА ПРАВОЙ ЧАСТИ

Р.Г. Идрисов, В.В. Коробицын

Предложен подход нахождения точки пересечения траектории решения системы обыкновенных дифференциальных уравнений с поверхностью разрыва правой части. В непрерывной области решение вычисляется с помощью численного метода Рунге–Кутты, а для нахождения точки пересечения применяется линейная интерполяция точек решения до и после пересечения поверхности разрыва. Результаты вычислительного эксперимента показали снижение глобальной погрешности численного решения по сравнению с методами, игнорирующими наличие разрыва.

Введение

Многие математические модели, описывающие процессы в механике, электротехнике, биологии и других областях, задаются системами обыкновенных дифференциальных уравнений (ОДУ) с разрывной правой частью [6], [12]. Траектория решения таких систем теряет гладкость на поверхности разрыва, что может приводить к росту погрешности численного решения при пересечении границы непрерывности.

Традиционно для нахождения численного решения обыкновенных дифференциальных уравнений используют методы Рунге–Кутты или Адамса, предполагая выполнение условия Липшица для функции правой части [1], [7], [8]. Однако для систем с разрывами применение этих методов приводит к неуправляемому росту погрешности. Даже если метод снабжен процедурой изменения шага интегрирования, все равно погрешность может увеличиваться в точке разрыва, и при этом может происходить рост числа итераций численного метода [2].

В настоящее время много работ посвящено численному решению систем ОДУ с разрывом в правой части [5], [9], [10], [11]. Однако, проведя обзор этих

Омский государственный университет им. Ф.М. Достоевского.
E-mail: idrisov@omskportal.ru

работ, нельзя сказать, что существуют универсальные алгоритмы решения систем ОДУ с разрывом в правой части.

В данной статье предлагается подход для дополнения традиционных методов Рунге–Кутты процедурой нахождения точки пересечения траектории решения с поверхностью разрыва. Дополнительные аспекты решения систем ОДУ с разрывами освещены в работах [3], [4].

1. Описание подхода вычисления точки разрыва

Рассмотрим систему обыкновенных дифференциальных уравнений вида

$$\frac{dx}{dt} = f(t,x),\ t>0,\ x(0)=x_0,$$

или в покомпонентной записи

$$\frac{dx_i}{dt} = f_i(t,x_1,...,x_n),\ t>0,$$

$$x_i(0) = x_i^0,\ i=1,2,...,n,$$

где x_i^0 — координаты начальной точки траектории решения $x_i(t)$. Некоторые функции $f_i(t,x_1,...,x_n)$ терпят разрыв на поверхностях, задаваемых уравнениями $\phi_j(t,x_1,...,x_n)=0$, $j=1,2,...k$.

Существует два подхода для численного решения систем ОДУ с разрывом в правой части: 1) игнорировать разрыв; 2) на поверхности разрыва остановить вычисление и продолжить с новыми значениями функций.

В первом случае при решении не учитывается особенность системы ОДУ с разрывом, и в точке пересечения кривой решения с поверхностью разрыва происходит увеличение погрешности до неприемлемых значений.

Для реализации второго подхода предлагаем следующий алгоритм:

а) вычисляем решение дифференциального уравнения шаг за шагом, пока одна из функций $\phi_j(t,x_1,...,x_n)$ не поменяет знак;

б) с помощью любого итерационного метода для решения нелинейных алгебраических уравнений (например, метода Ньютона) находим решение системы уравнений

$$\begin{cases} \phi_j(t,x_1,...,x_n) &= 0, \\ \psi(t,x_1,...,x_n) &= 0, \end{cases} \tag{1}$$

где $\psi(t,x_1,...,x_n)=0$ – уравнение прямой, составленной из точки решения до пересечения с поверхностью разрыва и точки решения после пересечения поверхности разрыва;

в) вычисляем решение дифференциального уравнения в найденной точке пересечения кривой решения с поверхностью разрыва и возобновляем вычисление уже с новыми значениями функций.

Для построения функции $\psi(t,x_1,...,x_n)$ необходимо взять точку решения $M_1(t^{(1)},x_1^{(1)},...,x_n^{(1)})$, полученную на последнем до пересечения с поверхностью

разрыва шаге, и точку $M_2(t^{(2)}, x_1^{(2)}, ..., x_n^{(2)})$ — первую точку решения, полученную после пересечения с поверхностью разрыва. Тогда уравнение прямой, проходящей через эти две точки, примет вид $\frac{t-t^{(1)}}{t^{(2)}-t^{(1)}} = \frac{x_1-x_1^{(1)}}{x_1^{(2)}-x_1^{(1)}} = ... = \frac{x_n-x_n^{(1)}}{x_n^{(2)}-x_n^{(1)}}$. Получаем

$$\psi(t, x_1, ..., x_n) = 0 \Leftrightarrow \begin{cases} \frac{x_1-x_1^{(1)}}{x_1^{(2)}-x_1^{(1)}} & = & \frac{t-t^{(1)}}{t^{(2)}-t^{(1)}}, \\ & ... & \\ \frac{x_n-x_n^{(1)}}{x_n^{(2)}-x_n^{(1)}} & = & \frac{t-t^{(1)}}{t^{(2)}-t^{(1)}}. \end{cases}$$

Решение этой системы можно выразить в явном виде

$$x_i(t) = x_i^{(1)} + \frac{x_i^{(2)}-x_i^{(1)}}{t^{(2)}-t^{(1)}}(t-t^{(1)}).$$

Тогда решение системы (1) преобразуется к виду

$$\phi_j(t, x_1(t), \ldots, x_n(t)) = 0,$$

где все $x_i(t)$ выражены явно через t. Задача вычисления точки пересечения кривой решения с поверхностью разрыва сводится к нахождению значения свободной переменной t, значение которой соответствует искомой точке. Однако в нашем случае эта точка будет показывать точку пересечения отрезка прямой линии, концы которого лежат в точках до и после пересечения поверхности разрыва.

2. Вычислительный эксперимент

Для тестирования описанного подхода для численного решения систем ОДУ с разрывом в правой части было исследовано поведение численных методов на решении следующей системы ОДУ:

$$\begin{cases} \frac{dx}{dt} = y - d_1, \\ \frac{dy}{dt} = x - c_1, \end{cases} \text{ при } x < \alpha,$$

$$\begin{cases} \frac{dx}{dt} = y - d_2, \\ \frac{dy}{dt} = x - c_2, \end{cases} \text{ при } x > \alpha, \qquad (2)$$

с заданными параметрами $d_1 = d_2 = 0.5, c_1 = 0.2, c_2 = 0.8, \alpha = 0.5$ и начальными данными $t_0 = 0, x_0 = 0.4999, y_0 = 0.3$.

Для этой системы известно точное решение:

$$x(t) = A_1 \cdot e^{t-t_0} + A_2 \cdot e^{-(t-t_0)} + a_x,$$
$$y(t) = A_1 \cdot e^{t-t_0} - A_2 \cdot e^{-(t-t_0)} + a_y,$$

где

$$A_1 = \frac{(x_0 - a_x) + (y_0 - a_y)}{2}, \quad A_2 = \frac{(x_0 - a_x) - (y_0 - a_y)}{2},$$

$$\begin{cases} a_x = c_1, \\ a_y = d_1, \end{cases} \text{при } x_0 < \alpha,$$

$$\begin{cases} a_x = c_2, \\ a_y = d_2, \end{cases} \text{при } x_0 > \alpha.$$

Решение системы терпит разрыв на прямой $x = \alpha$. Точное решение является периодическим с периодом

$$T = 2ln\left(\frac{|a_x - x_0| + \sqrt{(a_x - x_0)^2 - 4A_1A_2}}{2A_1}\right).$$

Траектория точного решения на фазовой плоскости изображена на рисунке 1.

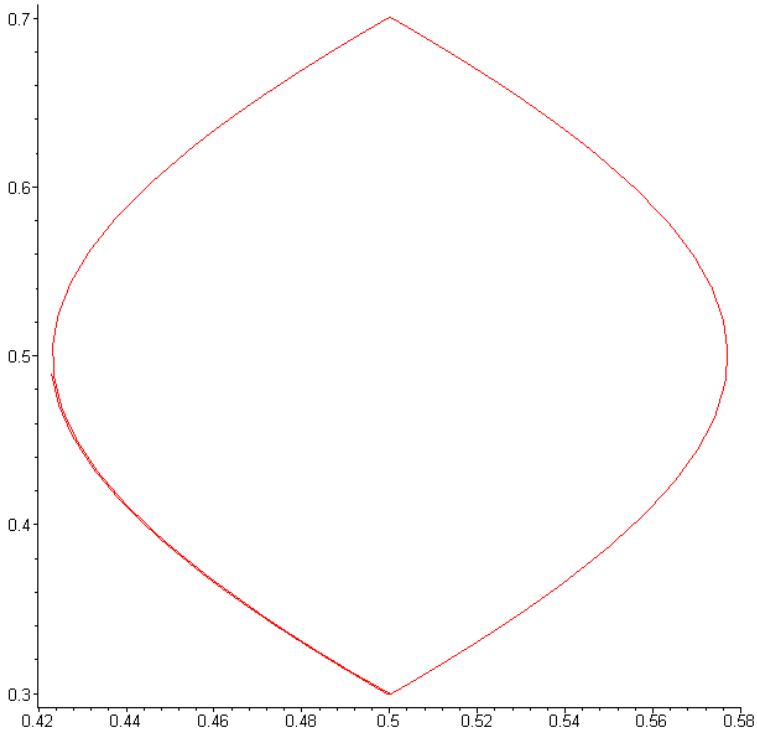


Рис. 1. Траектория решения системы (2) на фазовой плоскости

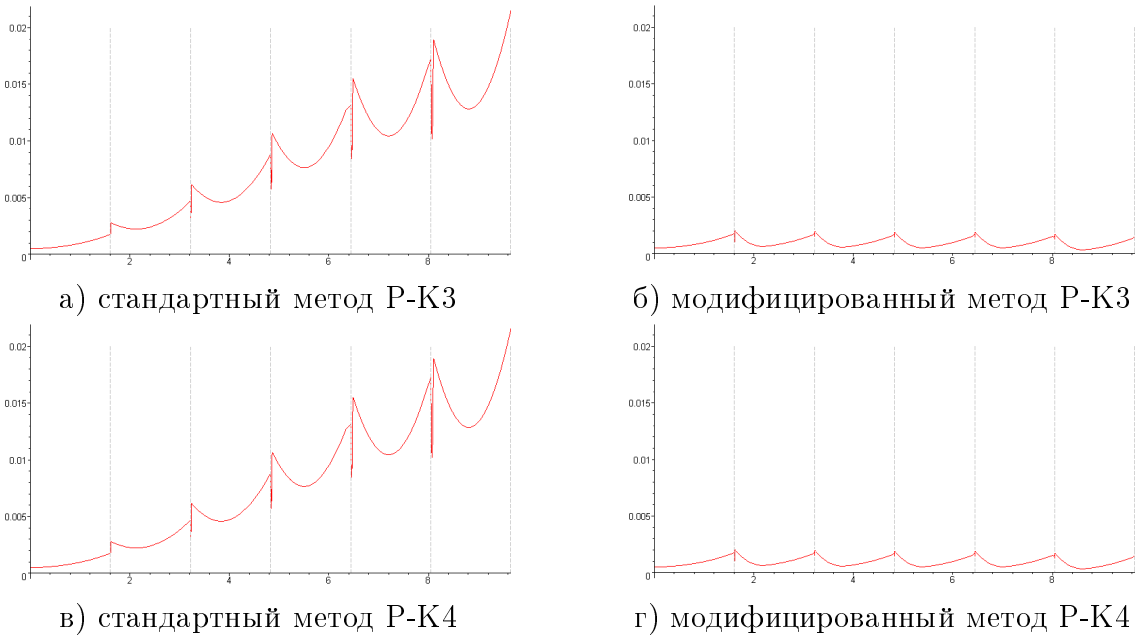
Для проведения эксперимента была написана программа на языке программирования C++, реализующая следующие методы решения систем ОДУ: методы Рунге—Кутты 3 и 4 порядка, метод Рунге—Кутты—Фельберга 4(5) порядка,

метод Розенброка (неявный метод Рунге—Кутты) и модификация вышеперечисленных методов для решения систем ОДУ с разрывом в правой части. Модификация заключалась в добавлении алгоритма вычисления точки пересечения траектории с поверхностью разрыва, описанном выше.

Для оценки эффективности применения модификации численных методов была проведена оценка локальной погрешности численного решения. Для этого в каждой точке траектории численного решения вычислялось значение локальной погрешности по формуле

$$Err(t_k) = \sqrt{(x_k - x^*(t_k))^2 + (y_k - y^*(t_k))^2},$$

где x_k, y_k — координаты точки численного решения, соответствующего моменту времени t_k, $x^*(t_k), y^*(t_k)$ — координаты точки точного решения в момент времени t_k. Графики изменения локальной погрешности при использовании методов Рунге—Кутты 3 и 4 порядков и соответствующих им модификаций представлены на рисунке 2. Как видно из рисунка, локальная погрешность стандартных методов растет с каждым периодом, в то время как погрешность модифицированного метода остается в пределах заданной точности.



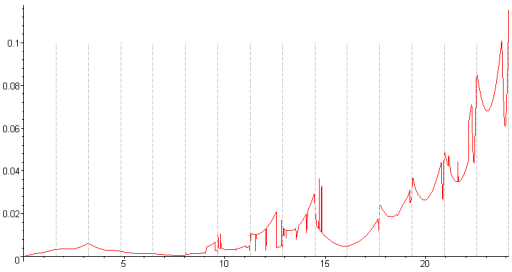
а) стандартный метод Р-К3 б) модифицированный метод Р-К3

в) стандартный метод Р-К4 г) модифицированный метод Р-К4

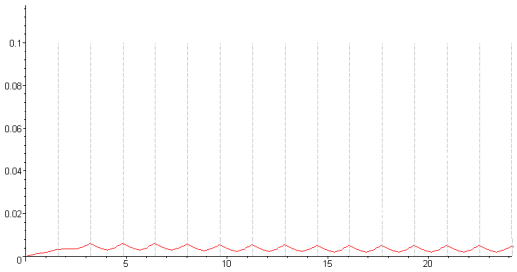
Рис. 2. Графики локальных погрешностей методов Рунге—Кутты 3 и 4 порядков до модификации и после (заданная точность 10^{-4})

Графики изменения локальной погрешности для методов, соответствующих схеме Рунге—Кутты—Фельберга 4 порядка с модификацией, представлены на рисунке 3. Из рисунка видно, что локальная погрешность метода до модификации растет, а после модификации метода остается в пределах заданной точности.

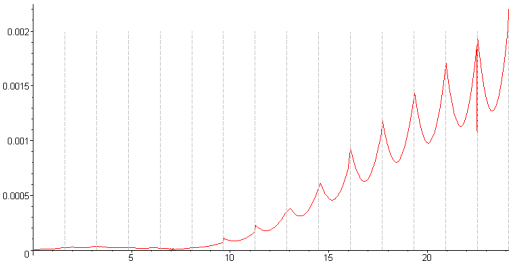
Графики изменения локальной погрешности метода Розенброка (неявный метод Рунге—Кутты) представлены на рисунке 4. Этот метод показал себя луч-



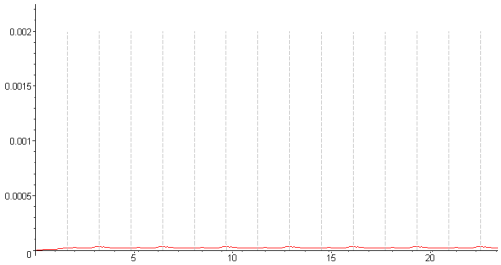
а) стандартный метод Р-К-Ф4 (заданная точность 10^{-4})



б) модифицированный метод Р-К-Ф4 (заданная точность 10^{-4})



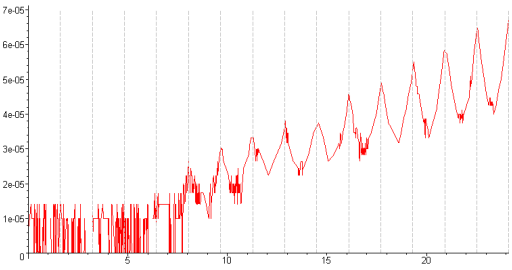
в) стандартный метод Р-К-Ф4 (заданная точность 10^{-8})



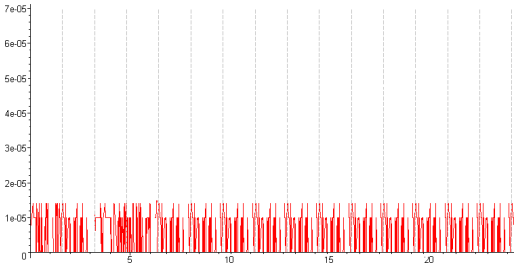
г) модифицированный метод Р-К-Ф4 (заданная точность 10^{-8})

Рис. 3. Графики локальных погрешностей методов Рунге—Кутты—Фельберга до модификации и после нее

ше других, поскольку неявный метод предполагает использование итерационной процедуры для поиска численного решения. Эта процедура оказывается очень полезной при поиске точки пересечения с поверхностью разрыва. Тем не менее для стандартного метода локальная погрешность все же растет, а после модификации метод дает очень хороший результат: локальная погрешность не превышает заданного порога точности.



а) стандартный метод Розенброка



б) модифицированный метод Розенброка

Рис. 4. Графики локальных погрешностей метода Розенброка до модификации и после нее

Делая общий обзор проведенного эксперимента, можно сказать, что использование предложенного алгоритма линейного приближения точки пересечения траектории решения ОДУ с поверхностью разрыва позволяет сохранить глобальную погрешность в пределах заданной точности, в то время как при игнорировании разрыва глобальная погрешность растет и превышает допустимую

через несколько периодов.

Таким образом, в представленной работе приведен способ уточнения точки пересечения кривой решения с поверхностью разрыва. В ходе реализации была решена задача по определению точки пересечения кривой решения с поверхностью разрыва. Также на примере одной системы проведен вычислительный эксперимент, позволяющий оценить точность решения систем ОДУ с разрывом в правой части при разных подходах. Представленный подход может быть использован для разработки программного продукта, предназначенного для численного исследования систем обыкновенных дифференциальных уравнений с разрывной правой частью.

Литература

1. Деккер К., Вервер Я. Устойчивость методов Рунге–Кутты для жестких нелинейных дифференциальных уравнений. М.: Мир, 1988. 334 с.
2. Коробицын В.В., Маренич В.Б., Фролова Ю.В. Исследование поведения явных методов Рунге-Кутты при решении систем обыкновенных дифференциальных уравнений с разрывной правой частью // Математические структуры и моделирование. 2007. Вып. 17. С. 19–25.
3. Коробицын В.В., Фролова Ю.В. Алгоритм численного решения дифференциальных уравнений с разрывной правой частью // Математические структуры и моделирование. 2005. Вып. 15. С. 46–54.
4. Коробицын В.В., Фролова Ю.В., Маренич В.Б. Алгоритм численного решения кусочно-сшитых систем // Вычислительные технологии. 2008. Т. 13, N. 2. С. 70-81.
5. Новиков Е.А., Шорников Ю.В. Численное моделирование гибридных систем методом Рунге—Кутты второго порядка точности // Вычислительные технологии. 2008. Т. 13, N. 2. С. 99–105.
6. Филиппов А. Ф. Дифференциальные уравнения с разрывной правой частью. М.: Наука, 1985. 225 с.
7. Хайрер Э., Ваннер Г. Решение обыкновенных дифференциальных уравнений. Жесткие и дифференциально-алгебраические задачи. М.: Мир, 1999. 685 с.
8. Хайрер Э., Нерсетт С., Ваннер Г. Решение обыкновенных дифференциальных уравнений: Нежесткие задачи. М.: Мир, 1990. 512 с.
9. Gear C.W., Osterby O. Solving ordinary differential equations with discontinuities // ACM Trans. Math. Software. 1984. Vol. 10. P. 23–44.
10. Piiroinen P.T., Kuznetsov Yu.A. An event-driven method to simulate Filippov systems with accurate computing of sliding motions // ACM Trans. Math. Software. 2008. Vol. 34, N. 13. P. 1–24.
11. Shampine L.F., Thompson S. Event location for ordinary differential equations // Computer and Mathematics with Application. 2000. Vol. 39. P. 43–54.
12. Zhusubaliyev Z., Mosekilde E. Bifurcations and chaos in piecewise-smooth dynamical systems. Singapore: World Scientific, 2003.

Математические структуры и моделирование
2010, вып. 21, с. 24–31

УДК 519.688

МНОГОМЕРНЫЕ ЛИНЕЙНЫЕ МНОГООБРАЗИЯ КАК СПОСОБ ВОССТАНОВЛЕНИЯ ГРАФИЧЕСКОЙ ИНФОРМАЦИИ

И.Б. Ларионов

Рассматривается метод восстановления графической информации с применением многомерных линейных многообразий.

Введение

В данной работе рассматривается метод восстановления графической информации с использованием многомерных линейных многообразий. Ранее в работе [2] рассматривался метод сингулярного разложения одномерным линейным многообразием.

Решение будет приведено явными формулами для двумерного и трехмерного случаев, а также будут приведены формулы для n-мерного случая.

1. Многомерные линейные многообразия

Для построения линейных многообразий размерности больше единицы используются совершенно аналогичные формулы. Но так как в составе одного многообразия размерности больше единицы, задающие его векторы могут быть не ортогональными, то для облегчения интерпретации полученных факторов рекомендуется проводить соответствующую ортогонализацию.

Рассмотрим двумерные и трехмерные линейные многообразия, для которых опишем процедуры построения и ортогонализации, а также приведем формулы для размерности n.

Однако хочется заметить, что, как будет показано ниже, использование многомерных линейных многообразий не принесет никакого выигрыша ни в качестве, ни в скорости в применении к восстановлению графической информации.

1.1. Ортогонализация базисной системы векторов

Пусть дано подпространство размерности n, и в нем система линейно независимых векторов $\{y_k\}$, $k = 1, ..., n$. По определению, они образуют базис этого

Омский государственный университет им. Ф.М. Достоевского.
E-mail: g0gich@mail.ws

подпространства. Требуется провести процедуру ортогонализации этого базиса, т.е. построить такой новый базис этого подпространства $\{\overline{y}_k\}(k = 1, ..., n)$, чтобы $(\overline{y}_j, \overline{y}_j) = 0$ для любых $i \neq j$.

Для начала решим следующую задачу.

Пусть набор векторов $\{y_k\}(k = 1, ..., n-1)$ уже ортогонализирован, а y_n – еще нет. Тогда построим новый вектор $\overline{y}_n = \sum_{i=1}^{n-q} \alpha_i y_i + y_n$ и потребуем, чтобы $(\overline{y}_n, y_k) = 0$ для всех $k = 1, ..., n-1$. В результате имеем систему из $n-1$ уравнения с $n-1$ неизвестными:

$$\big(\sum_{i=1}^{n} \alpha_i y_i + y_n, y_k\big) = 0, k = 1, ..., n-1.$$

Учитывая, что $(y_i, y_k) = 0$ при $i \neq k$, то легко заметить, что полученная система будет диагональной. Отсюда неизвестные легко находятся по формуле

$$\alpha_k = -\frac{(y_n, y_k)}{y_k, y_k}.$$

Следовательно, если задана система из n векторов, в которой имеется ортогональная подсистема из $n-1$ вектора, то оставшийся вектор «ортогонализируется» по формуле

$$\overline{y}_n = y_n - \sum_{i=1}^{n-1} \frac{(y_n, y_i)}{(y_i, y_i)} y_i. \tag{1}$$

Тогда процедура ортогонализации системы из n базисных векторов выглядит как последовательная ортогонализация систем из $2, 3, \ldots, n$ векторов с использованием формулы 1.

1.2. Двумерные линейные модели

Для построения двумерного линейного многообразия минимизируем квадратичную форму:

$$\Phi = \sum_{\substack{i,j \\ a_{ij} \neq @}} (a_{ij} - x_{1i} y_{1j} - x_{2i} y_{2j} - b_j) \to \min, \tag{2}$$

где @ – пропущенные элементы. Решение дается последовательными итерациями по явным формулам. При фиксированных y_{1j}, y_{2j} и b_j значения x_{1i} и x_{2i} однозначно находятся из системы равенств $\partial\Phi/\partial x_{1i} = 0$ и $\partial\Phi/\partial x_{2i} = 0$:

$$\begin{cases} \partial\Phi/\partial x_{1i} = -2 \sum_{j,\ a_{ij} \neq @} (a_{ij} - x_{1i} y_{1j} - x_{2i} y_{2j} - b_j) y_{1j} = 0, \\ \partial\Phi/\partial x_{2i} = -2 \sum_{j,\ a_{ij} \neq @} (a_{ij} - x_{1i} y_{1j} - x_{2i} y_{2j} - b_j) y_{2j} = 0. \end{cases}$$

То же самое в векторном виде:

$$\begin{cases} x_{1i}(y_1, y_1)_{a_j} + x_{2i}(y_2, y_1)_{a_i} = (a_i - b, y_1)_{a_i}, \\ x_{1i}(y_1, y_2)_{a_j} + x_{2i}(y_2, y_2)_{a_i} = (a_i - b, y_2)_{a_i}. \end{cases}$$

Аналогично при фиксированных x_{1i} и x_{2i} значения, доставляющие минимум квадратичной форме 2, однозначно находятся из равенств $\partial\Phi/\partial y_{1j} = 0$, $\partial\Phi/\partial y_{2j} = 0$ и $\partial\Phi/\partial b_j = 0$:

$$\begin{cases} \partial\Phi/\partial y_{1j} = -2\sum_{i,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - b_j)x_{1i} = 0, \\ \partial\Phi/\partial y_{2j} = -2\sum_{i\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - b_j)x_{2i} = 0, \\ \partial\Phi/\partial b_j = -2\sum_{i,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - b_j) = 0. \end{cases}$$

То же самое в векторной форме:

$$\begin{cases} y_{1j}(x_1, x_1)_{a_j} + y_{2j}(x_2, x_1)_{a_j} + b_j(x_1, 1)_{a_j} = (a_j, x_1)_{a_j}, \\ y_{1j}(x_1, x_2)_{a_j} + y_{2j}(x_2, x_2)_{a_j} + b_j(x_2, 1)_{a_j} = (a_j, x_2)_{a_j}, \\ y_{1j}(x_1, 1)_{a_j} + y_{2j}(x_2, 1)_{a_j} + b_j(1, 1)_{a_j} = (a_j, 1)_{a_j}. \end{cases}$$

Полученная система решается любым численным способом, например методом квадратного корня (т.к. полученная матрица является симметричной). Учитывая, что полученные векторы y_1 и y_2 могут быть не ортогональны, то в некоторых случаях их необходимо ортогонализировать. А так как они образуют базис подпространства размерности 2, то эта задача легко решается с использованием формулы 1.

1.3. Трехмерные линейные модели

Для построения трехмерного линейного многообразия минимизируем квадратичную форму:

$$\Phi = \sum_{\substack{i,j \\ a_{ij}\neq@}} (a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - x_{3i}y_{3j} - b_j)^2 \to \min. \tag{3}$$

Решение дается последовательными итерациями по явным формулам. При фиксированных y_{1j}, y_{2j}, y_{3j} и b_j значения x_{1i}, x_{2i} и x_{3i} однозначно находятся из системы равенств $\partial\Phi/\partial x_{1i} = 0$, $\partial\Phi/\partial x_{2i} = 0$ и $\partial\Phi/\partial x_{3i} = 0$:

$$\begin{cases} \partial\Phi/\partial x_{1i} = -2\sum_{i,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - x_{3i}y_{3j} - b_j)y_{1j} = 0, \\ \partial\Phi/\partial x_{2i} = -2\sum_{i,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - x_{3i}y_{3j} - b_j)y_{2j} = 0, \\ \partial\Phi/\partial x_{3i} = -2\sum_{i,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - x_{3i}y_{3j} - b_j)y_{3j} = 0. \end{cases}$$

То же самое в векторной форме:

$$\begin{cases} x_{1i}(y_1, y_1)_{a_j} + x_{2i}(y_2, y_1)_{a_j} + x_{3i}(y_3, y_1)_{a_j} = (a_j - b, y_1)_{a_j}, \\ x_{1i}(y_1, y_2)_{a_j} + x_{2i}(y_2, y_2)_{a_j} + x_{3i}(y_3, y_2)_{a_j} = (a_j - b, y_2)_{a_j}, \\ x_{1i}(y_1, y_3)_{a_j} + x_{2i}(y_2, y_3)_{a_j} + x_{3i}(y_3, y_3)_{a_j} = (a_j - b, y_3)_{a_j}. \end{cases}$$

Аналогично при фиксированных x_{1i}, x_{2i} и x_{3i} значения, доставляющие минимум квадратичной форме 3, однозначно находятся из равенств $\partial\Phi/\partial y_{1j} = 0$, $\partial\Phi/\partial y_{2j} = 0$, $\partial\Phi/\partial y_{3j} = 0$ и $\partial\Phi/\partial b_j = 0$:

$$\begin{cases} \partial\Phi/\partial y_{1j} = -2\sum_{i,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - x_{3i}y_{3j} - b_j)x_{1i} = 0, \\ \partial\Phi/\partial y_{2j} = -2\sum_{i,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - x_{3i}y_{3j} - b_j)x_{2i} = 0, \\ \partial\Phi/\partial y_{2j} = -2\sum_{i,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - x_{3i}y_{3j} - b_j)x_{2i} = 0, \\ \partial\Phi/\partial b_j = -2\sum_{i,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - x_{3i}y_{3j} - b_j) = 0. \end{cases}$$

То же самое в векторной форме:

$$\begin{cases} y_{1j}(x_1, x_1)_{a_j} + y_{2j}(x_2, x_1)_{a_j} + y_{3j}(x_3, x_1)_{a_j} + b_j(x_1, 1)_{a_j} = (a_j - b, x_1)_{a_j}, \\ y_{1j}(x_1, x_2)_{a_j} + y_{2j}(x_2, x_2)_{a_j} + y_{3j}(x_3, x_2)_{a_j} + b_j(x_2, 1)_{a_j} = (a_j - b, x_2)_{a_j}, \\ y_{1j}(x_1, x_3)_{a_j} + y_{2j}(x_2, x_3)_{a_j} + y_{3j}(x_3, x_3)_{a_j} + b_j(x_3, 1)_{a_j} = (a_j - b, x_3)_{a_j}, \\ y_{1j}(x_1, 1)_{a_j} + y_{2j}(x_2, 1)_{a_j} + y_{3j}(x_3, 1)_{a_j} + b_j(1, 1)_{a_j} = (a_j - b, 1)_{a_j}. \end{cases}$$

Полученная система решается любым численным способом, например методом квадратного корня (т.к. полученная матрица является симметричной).

Учитывая, что полученные векторы y_1, y_2 и y_3 могут быть не ортогональны, то в некоторых случаях их необходимо ортогонализировать. А так как они образуют базис подпространства размерности 3, то эта задача легко решается с использованием формулы 1.

1.4. Линейная модель произвольной размерности

Для построения линейного многообразия произвольной размерности минимизируем квадратичную форму:

$$\Phi = \sum_{\substack{i,j \\ a_{ij}\neq@}} (a_{ij} - \sum_n x_{ni}y_{nj} - b_j)^2 \to \min, \tag{4}$$

где n - размерность линейного многообразия.

Решение дается последовательными итерациями по явным формулам. При фиксированных y_{nj} и b_j значения x_{ni} однозначно находятся из системы равенств $\partial\Phi/\partial x_{ni} = 0$:

$$\begin{cases} \partial\Phi/\partial x_{1i} = -2\sum_{j,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - \cdots - x_{ni}y_{nj} - b_j)y_{1j} = 0, \\ \partial\Phi/\partial x_{2i} = -2\sum_{j,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - \cdots - x_{ni}y_{nj} - b_j)y_{2j} = 0, \\ \ldots \\ \partial\Phi/\partial x_{ni} = -2\sum_{j,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - \cdots - x_{ni}y_{nj} - b_j)y_{nj} = 0. \end{cases}$$

То же самое в векторном виде:

$$\begin{cases} x_{1i}(y_1, y_1)_{a_j} + x_{2i}(y_2, y_1)_{a_i} + \cdots + x_{ni}(y_n, y_1)_{a_i} = (a_i - b, y_1)_{a_i}, \\ x_{1i}(y_1, y_2)_{a_j} + x_{2i}(y_2, y_2)_{a_i} + \cdots + x_{ni}(y_n, y_2)_{a_i} = (a_i - b, y_2)_{a_i}, \\ \ldots \\ x_{1i}(y_1, y_n)_{a_j} + x_{2i}(y_2, y_n)_{a_i} + \cdots + x_{ni}(y_n, y_n)_{a_i} = (a_i - b, y_n)_{a_i}. \end{cases}$$

Аналогично при фиксированных x_{ni} значения, доставляющие минимум квадратичной форме 4, однозначно находятся из равенств $\partial\Phi/\partial y_{nj} = 0$ и $\partial\Phi/\partial b_j = 0$:

$$\begin{cases} \partial\Phi/\partial y_{1j} = -2\sum_{i,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - \cdots - x_{ni}y_{nj} - b_j)x_{1i} = 0, \\ \partial\Phi/\partial y_{2j} = -2\sum_{i,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - \cdots - x_{ni}y_{nj} - b_j)x_{2i} = 0, \\ \cdots \\ \partial\Phi/\partial y_{nj} = -2\sum_{i,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - \cdots - x_{ni}y_{nj} - b_j)x_{ni} = 0, \\ \partial\Phi/\partial b_j = -2\sum_{i,\ a_{ij}\neq@}(a_{ij} - x_{1i}y_{1j} - x_{2i}y_{2j} - \cdots - x_{ni}y_{nj} - b_j) = 0. \end{cases}$$

То же самое в векторной форме:

$$\begin{cases} y_{1j}(x_1, x_1)_{a_j} + y_{2j}(x_2, x_1)_{a_j} + \cdots + y_{nj}(x_n, x_1)_{a_j} + b_j(x_1, 1)_{a_j} = (a_j, x_1)_{a_j}, \\ y_{1j}(x_1, x_2)_{a_j} + y_{2j}(x_2, x_2)_{a_j} + \cdots + y_{nj}(x_n, x_2)_{a_j} + b_j(x_2, 1)_{a_j} = (a_j, x_2)_{a_j}, \\ \cdots \\ y_{1j}(x_1, x_n)_{a_j} + y_{2j}(x_2, x_n)_{a_j} + \cdots + y_{nj}(x_n, x_n)_{a_j} + b_j(x_n, 1)_{a_j} = (a_j, x_n)_{a_j}, \\ y_{1j}(x_1, 1)_{a_j} + y_{2j}(x_2, 1)_{a_j} + \cdots + y_{nj}(x_n, 1)_{a_j} + b_j(1, 1)_{a_j} = (a_j, 1)_{a_j}. \end{cases}$$

Полученная система решается любым численным способом, например методом квадратного корня (т.к. полученная матрица является симметричной). Учитывая, что полученные векторы y_n могут быть не ортогональны, то в некоторых случаях их необходимо ортогонализировать. А так как они образуют базис подпространства размерности n, то эта задача легко решается с использованием формулы 1.

2. Применение многомерных линейных многообразий для восстановления графической информации

В ходе работы был реализован класс MatrixSolver, который при заданной размерности многообразия строил вектора x_n, y_n и b и производил итерационное исчерпание матрицы данных, как это описано в работе [2].

Итерационные вычисления проводились согласно приведенным формулам многомерных линейных многообразий, приведенных в векторной форме. Данный подход позволил строить СЛАУ, решение которых не вызывало серьезных технических и временных затрат. Все вычисления на каждой итерации сводились к построению и решению i СЛАУ порядка n и j СЛАУ порядка $n + 1$.

Были проведены испытания для размерностей 2, 3, ..., 10. Исходя из полученных на тот момент результатов были проведены испытания для размерности 100. Все испытания проводились с одним и тем же изображением, имеющем размеры 1024x768 пикселей, что позволяет судить о количестве решенных СЛАУ и их размерности.

Оценка ошибки производилось методом среднего квадратичного [4]. Данная метрика была выбрана как наиболее наглядная и простая для вычисления. Кроме того, в рамках данной работы ключевым моментом является выявление зависимости размерности линейного многообразия и таких его свойств, как время, затраченное на достижение определенной, наперед заданной максимальной ошибки.

Для начала в таблице 1 приведем зависимости ошибки восстановления на первых 10-ти шагах восстановления для разных размерностей.

Таблица 1. Ошибка восстановления при разных размерностях

i/n	2	3	4	5	6
1	250691595,68	113356199,79	64534470,18	41384009,45	27736091,44
2	125345797,84	56678099,89	32267235,09	20692004,72	13868045,72
3	83563865,23	37785399,93	21511490,06	13794669,82	9245363,81
4	62672898,92	28339049,95	16133617,54	10346002,36	6934022,86
5	50138319,14	22671239,96	12906894,04	8276801,89	5547218,29
6	41781932,61	18892699,96	10755745,03	6897334,91	4622681,91
7	35813085,10	16193742,83	9219210,03	5912001,35	3962298,78
8	31336449,46	14169524,97	8066808,77	5173001,18	3467011,43
9	27854621,74	12595133,31	7170496,69	4598223,27	3081787,94
10	25069159,57	11335619,98	6453447,02	4138400,94	2773609,14
i/n	7	8	9	10	100
1	20368692,15	15482141,30	10264537,78	3636251,88	51516,38
2	10184346,07	7741070,65	5132268,89	1818125,94	25758,19
3	6789564,05	5160713,77	3421512,59	1212083,96	17172,13
4	5092173,04	3870535,33	2566134,44	909062,97	12879,10
5	4073738,43	3096428,26	2052907,56	727250,38	10303,28
6	3394782,02	2580356,88	1710756,30	606041,98	8586,06
7	2909813,16	2211734,47	1466362,54	519464,55	7359,48
8	2546086,52	1935267,66	1283067,22	454531,48	6439,55
9	2263188,02	1720237,92	1140504,20	404027,99	5724,04
10	2036869,21	1548214,13	1026453,78	363625,19	5151,64

Далее для наглядности на рисунке 1 приведем график среднеквадратичной ошибки на первых 10-ти итерациях для $n = 2, \ldots, 10, 100$.

В ходе испытаний было выявлено минимальное значение ошибки, которого удалось достичь при восстановлении изображений - 4760. После достижения данного значения при всех рассматриваемых n алгоритм останавливался, так как межшаговое падение ошибки было достаточно малым [2].

Приведем таблицу длительности одной итерации.

Таблица 2. Длительность первой итерации

n	2	3	4	5	6	7	8	9	10	100
Длительность, мс	52	115	202	315	470	640	842	1270	3585	153045

Как видно из этой таблицы, длительность одной итерации пропорциональна квадрату размерности многообразия, что говорит о малом количестве накладных расходов на построение матриц и их решение.

Для оценки эффективности применения линейных многообразий больших порядков рассмотрим изменение ошибки восстановления за единицу времени.

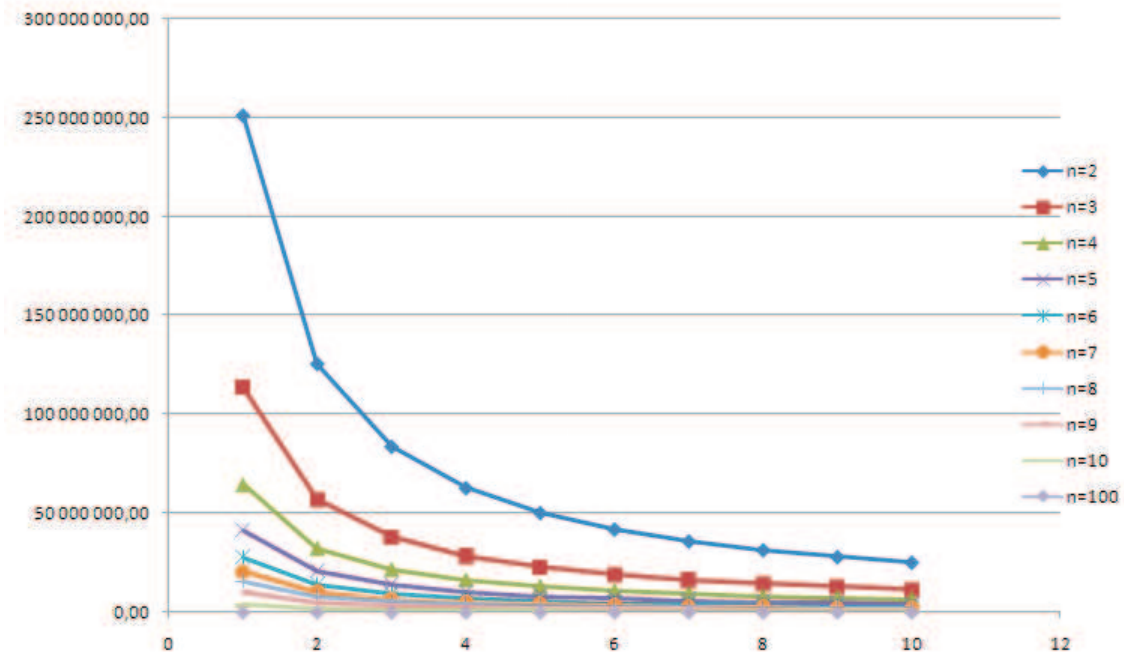


Рис. 1. График среднеквадратичной ошибки на первых 10-ти итерациях

Вычисления производятся по формуле

$$\partial\Phi_i = \frac{\Phi_{i-1} - \Phi_i}{\Delta t},$$

где Φ_i - ошибка восстановления на i-й итерации, а Δt - время выполнения итерации. Сведем данные в таблицу.

Как видно из таблицы 3, эффективность падает не только от итерации к итерации, но и с ростом размерности n. В таблице 4 приведено время, требуемое для остановки алгоритма согласно критериям остановки для каждой размерности.

Выводы

В ходе работы были выведены формулы построения линейного многообразия произвольной размерности и произведены испытания последовательного исчерпания матрицы с пропусками.

Результаты испытаний наглядно показывают, что данный метод может применяться для восстановления графической информации, однако использование больших размерностей не приводит ни к ускорению восстановления, ни к улучшению качества восстановления.

Кроме того, увеличение размерности линейного многообразия влечет за собой только увеличение длительности выполнения алгоритма, а также уменьшение количества итераций, требуемых для остановки алгоритма. Однако во всех рассмотренных случаях алгоритм останавливался из-за малого падения ошибки за итерацию, и во всех случаях значение конечной ошибки было не менее 5000.

Таблица 3. Временная эффективность алгоритма

i/n	2	3	4	5	6
2	2410496,11	492853,04	159738,79	65688,90	29506,48
3	803498,70	164284,35	53246,26	21896,30	9835,49
4	401749,35	82142,17	26623,13	10948,15	4917,75
5	241049,61	49285,30	15973,88	6568,89	2950,65
6	160699,74	32856,87	10649,25	4379,26	1967,10
7	114785,53	23469,19	7606,61	3128,04	1405,07
8	86089,15	17601,89	5704,96	2346,03	1053,80
9	66958,23	13690,36	4437,19	1824,69	819,62
10	53566,58	10952,29	3549,75	1459,75	655,70
i/n	7	8	9	10	100
2	15913,04	9193,67	4041,16	507,15	0,10
3	5304,35	3064,56	1347,05	169,05	0,03
4	2652,17	1532,28	673,53	84,52	0,02
5	1591,30	919,37	404,12	50,71	0,01
6	1060,87	612,91	269,41	33,81	0,01
7	757,76	437,79	192,44	24,15	0,00
8	568,32	328,35	144,33	18,11	0,00
9	442,03	255,38	112,25	14,09	0,00
10	353,62	204,30	89,80	11,27	0,00

Таблица 4. Время работы алгоритма, t MM:CC

n	2	3	4	5	6	7	8	9	10	100
t	11:53	11:54	11:54	11:54	11:55	11:55	11:56	11:59	12:05	15:23

Из всего вышесказанного можно сделать вывод, что восстановление изображений при помощи итерационного исчерпания матрицы с пропусками с использованием линейных многомерных многообразий следует проводить только в двумерном случае.

Литература

1. Горбань А.Н., Макаров С.В., Россиев А.А. Итерационный метод главных компонент для таблиц с пробелами // Третий сибирский конгресс по прикладной и индустриальной математике (ИНПРИМ-98), 22–27 июня 1998. Тезисы докладов. Ч. 5. Новосибирск: Изд-во Института математики СО РАН, 1998.
2. Ларионов И.Б. Кластеризация матриц с пропусками как метод восстановления графической информации // Математические структуры и моделирование. 2009. Вып. 20. С. 97–106.
3. Самарский А.А. Введение в численные методы М.: Наука, 1982.
4. Ismail Avcibas, Ismail Avcibas, Bulent Sankur, Lale Akarun, Emin Anarim, Nasir Memon, Yucel Yemez. Image Quality Statistics and Their Use in Steganalysis and Compression. 2001.

Математические структуры и моделирование
2010, вып. 21, с. 32–36

УДК 519.245

ТЕСТИРОВАНИЕ ГЕНЕРАТОРОВ ПСЕВДОСЛУЧАЙНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ С ИСПОЛЬЗОВАНИЕМ ТРЕХМЕРНОЙ МОДЕЛИ ИЗИНГА

А.Ю. Шерешик

В статье исследуются реализации трехмерной модели Изинга методом Монте-Карло, где в роли генератора случайных чисел используются различные генераторы псевдослучайных последовательностей. Проводится анализ влияния уравнения и структуры генератора на отклонение результатов моделирования от теоретических расчетов.

Введение

В качестве метода численного статистического моделирования в настоящее время широко применяется метод Монте-Карло. В частности, для решения ряда различных задач физики, химии, биологии, кибернетики и др. применяется компьютерный эксперимент. При этом обеспечение достаточной точности во многом зависит от правильного выбора уравнения и структуры генератора случайных чисел. Также существует и обратная зависимость: с помощью известных результатов теоретических расчетов мы можем оценить, насколько достоверно тот или иной генератор моделирует физический процесс.

В подобном эксперименте полезно также рассматривать результаты с точки зрения продолжительности расчетов. Таким образом, с использованием трехмерной модели Изинга можно провести тестирование качества и скорости генераторов псевдослучайных последовательностей.

В представленной работе проводится исследование влияния выбора генератора случайных чисел на отклонение параметров модели от рассчитанных теоретически. В ходе эксперимента рассмотрены различные конгруэнтные генераторы случайных чисел. Проводится моделирование, анализ и сравнение поведения модели при температуре, близкой к критической. Делаются выводы о точности моделирования фазового перехода.

Омский государственный университет им. Ф.М. Достоевского.
E-mail: shereshik@gmail.com

1. Описание эксперимента

Задача эксперимента – найти, какие из рассмотренных генераторов позволяют, за достаточно короткий промежуток времени, достоверно моделировать трехмерную модель Изинга. Таким образом, оценка результатов производилась по двум параметрам: продолжительность и качество моделирования. Качество оценивалось по графику зависимости средней теплоемкости и магнитной восприимчивости от температуры. График должен иметь явно выраженный пик в области температуры фазового перехода. По данным [3, 6] для трехмерной модели Изинга с взаимодействием шести спинов температура фазового перехода kT_c/J находится в интервале от 4.5103 до 4.53.

Для сравнения были рассмотрены подобные работы, посвященные исследованию двухмерной модели Изинга методом Монте-Карло. Опубликованные результаты позволяют утверждать, что для моделирования на двухмерной решетке некоторые из рассмотренных генераторов показали хорошие результаты при количестве шагов порядка 10^7 и линейном размере решетки 16 [1].

В нашем случае гораздо более важно то, какой из генераторов показывает приемлемые результаты при малом количестве шагов – так как реализация трехмерной модели значительно более затратна по машинному времени.

2. Генераторы случайных чисел

2.1. Линейный конгруэнтный генератор

Линейный конгруэнтый генератор в общем случае имеет вид:

$$x_{n+1} = (ax_n + b) \mod m,$$

где структуру определяют модуль – m, множитель – a, приращение – b и начальное значение – x_0. Это наиболее простой из рассмотренных генераторов. Подобные генераторы обладают довольно хорошими статистическими свойствами. Значения, приведенные в работе [5], были выбраны для определения структуры. В результате уравнение рассматриваемого нами генератора выглядит так:

$$x_{n+1} = (2416x_n + 374441) \mod 1771875.$$

2.2. Инверсивный конгруэнтый генератор

Инверсивный конгруэнтый генератор в общем случае имеет вид:

$$x_n = (a\overline{x}_{n-1} + b) \mod m,$$

где m–простое число, $x^{-1}x = 1 \mod m$. Подобные генераторы обладают очень большим периодом, равным m. Однако, в связи со сложным алгоритмом вычисления числа, обратного по модулю, можно предположить, что этот генератор окажется самым медленным из рассмотренных нами генераторов. Значения, приведенные в работе [2], были выбраны для определения структуры. В результате уравнение рассматриваемого нами генератора выглядит таким образом:

$$x_n = (1208490188\overline{x}_{n-1} + 1) \mod 2147483647.$$

2.3. Генератор Фибоначчи с запаздыванием

Генераторы Фибоначчи становятся все более популярны, поскольку предлагают простой метод получения очень больших перодов. Генератор в общем случае имеет вид:

$$x_n = (x_{n-p} \odot x_{n-q}),$$

где $\odot$ – одна из простых бинарных арифметических операций: $+, -, *, \oplus$. В работе [1] были отмечены и рекомендованы для использования при моделировании двухмерной решетки Изинга некоторые из подобных генераторов. Мы рассмотрим четыре из них:

$$x_n = (x_{n-250} + x_{n-103});\ x_n = (x_{n-1279} + x_{n-1063});$$

$$x_n = (x_{n-250} \oplus x_{n-103});\ x_n = (x_{n-1279} \oplus x_{n-1063}).$$

Следует также отметить, что при увеличении индексов p, q увеличивается период, но ощутимо замедляется скорость работы генератора. Это связано с тем, что для хранения списка предыдущих элементов требуется все больше операций с памятью.

3. Результаты эксперимента

Эксперимент проводился на компьютерной модели трехмерной решетки Изинга с линейным размером 16. Вычисление параметров средней теплоемкости и восприимчивости системы проводилось на интервале температур от 4.4 до 4.6 kT_c/J с шагом 0.01. Число отбрасываемых шагов для установления релаксации – 100 шагов на спин. Время эксперимента – 10000 шагов на спин.

Рассмотрим результаты по контрольным параметрам.

3.1. Скорость генераторов

Таблица 1. Продолжительность моделирования в зависимости от выбора генератора

$x_{n+1} = (2416x_n + 374441) \mod 1771875$	2 ч 51 мин 35,223 с
$x_n = (1208490188\overline{x}_{n-1} + 1) \mod 2147483647$	31 ч 48 мин 21,392 с
$x_n = (x_{n-250} + x_{n-103})$	6 ч 2 мин 50,173 с
$x_n = (x_{n-250} \oplus x_{n-103})$	5 ч 10 мин 37,805 с
$x_n = (x_{n-1279} + x_{n-1063})$	14 ч 23 мин 0,297 с
$x_n = (x_{n-1279} \oplus x_{n-1063})$	12 ч 56 мин 33,367 с

Можно заметить, насколько сильно влияет выбор уравнения на продолжительность моделирования (табл. 1). Также видно, что инверсивный генератор вряд ли можно считать эффективным с точки зрения затрат машинного времени. Очевидно, это связано со сложностью задачи нахождения числа, обратного по модулю.

3.2. Качество генераторов

Оценка точности имитации физического процесса проводилась по графикам функций зависимости контрольных параметров от времени. Результат эксперимента будем считать положительным, если график имеет только один выраженный пик в области поиска температуры фазового перехода.

Четыре из шести рассмотренных генераторов, при выбранных параметрах эксперимента, дали отрицательный результат. Значения магнитной восприимчивости изменяются достаточно хаотично либо более одного выраженного пика (рис. 1).

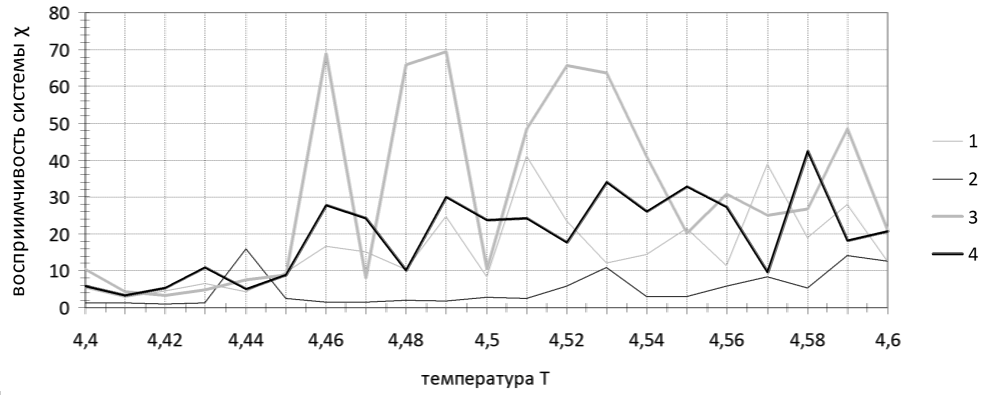


Рис. 1. График зависимости намагниченности от температуры. Отрицательные результаты ($1 - x_{n+1} = (2416x_n + 374441) \mod 1771875$; $2 - x_n = (1208490188\overline{x}_{n-1} + 1) \mod 2147483647$; $3 - x_n = (x_{n-250} + x_{n-103})$; $4 - x_n = (x_{n-250} \oplus x_{n-103})$)

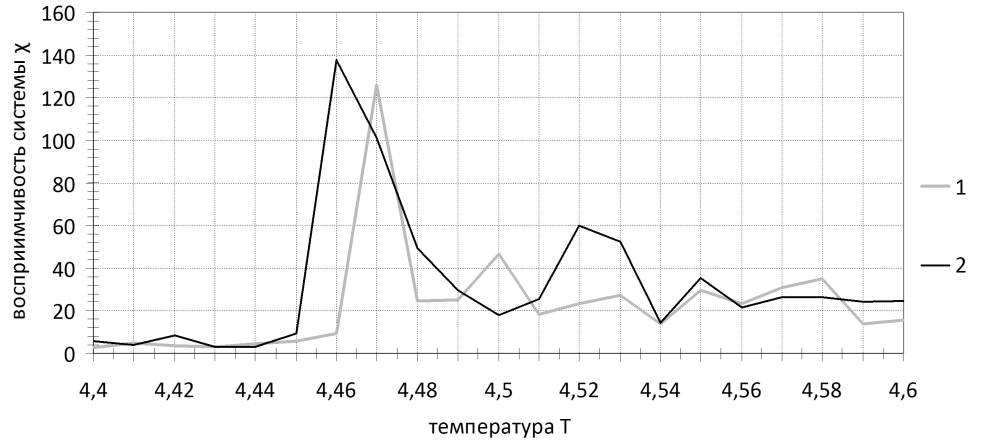


Рис. 2. График зависимости намагниченности от температуры. Положительные результаты. $1 - x_n = (x_{n-1279} + x_{n-1063})$; $2 - x_n = (x_{n-1279} \oplus x_{n-1063})$

Положительный результат был получен при выборе генератора Фибоначчи с запаздыванием с большими коэффициентами p и q. На графике четко видны пики в обоих случаях (рис. 2). Также можно отметить, что пики видны и на графике изменения средней теплоемкости, однако температура фазового перехода несколько отличается от показателей при измерении намагниченности (рис. 3).

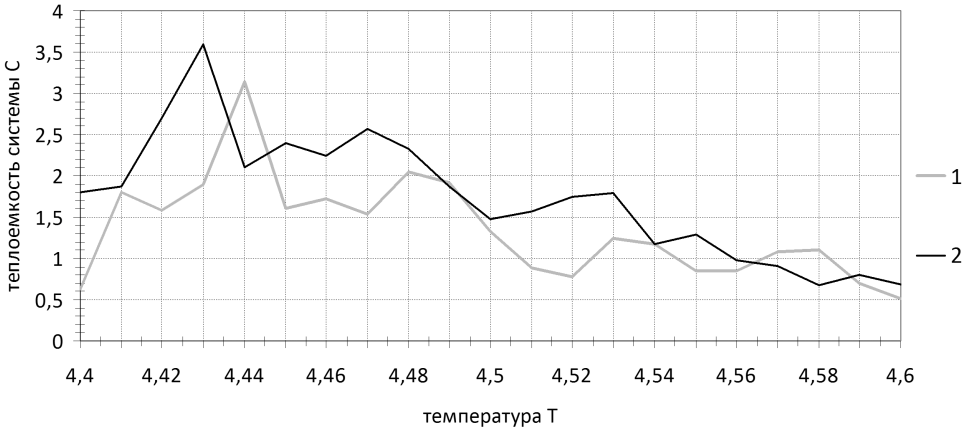


Рис. 3. График зависимости теплоемкости от температуры. Положительные результаты. $1 - x_n = (x_{n-1279} + x_{n-1063})$; $2 - x_n = (x_{n-1279} \oplus x_{n-1063})$

3.3. Результат

По результатам эксперимента наиболее подходящим можно считать генератор, имеющий уравнение $x_n = (x_{n-1279} \oplus x_{n-1063})$. Из двух положительных результатов этот был достигнут быстрее за счет лежащей в его основе менее сложной банарной операции. При достаточно хорошем качестве моделирования скорость работы этого алгоритма более чем в 2 раза выше, чем скорость инверсивного генератора.

Литература

1. Coddington P.D. Tests of random number generators using Ising model simulations // Int. J. Modern Physics C. 1996. V.7, N. 3, P. 295–303.
2. Hellekalek P. Inversive pseudorandom number generators: concepts, results, and links // Winter Simulation Conference (WSC'95). 1995. P. 255–262.
3. Вакилов А.Н., Марков О.Н., Прудников В.В. Компьютерное моделирование фазовых переходов в однородных и неупорядоченных системах. Учебное пособие. Омск.: Изд-во ОмГУ, 2001. 85 с.
4. Гулд Х., Тобочник Я. Компьютерное моделирование в физике: В 2-х частях. Ч. 2: Пер. с англ. М.: Мир, 1990. 400 с.
5. Иванов М.А., Чугунков И.В. Теория, применение и оценки качества генераторов псевдослучайных последовательностей. М.: Изд-во Кудиц-образ, 2003. 238 с.
6. Нигматуллин Р. Р., Тобоев В. А. Термодинамика основных трехмерных моделей ферромагнетиков во флуктуационном приближении // ТМФ. 1988. Т.74, N. 1. С.112–124.

Математические структуры и моделирование
2010, вып. 21, с. 37–43

УДК 004.3

РЕАЛИЗАЦИЯ ГЕНЕРАТОРА СЛУЧАЙНЫХ ЧИСЕЛ НА БАЗЕ ЗВУКОВОЙ КАРТЫ

Д.Б. Беспалов, С.В. Белим

Работа посвящена созданию аппаратного генератора случайных последовательностей на основе интегрированной звуковой карты. Проводится тестирование генерируемых последовательностей.

Введение

Потребности в последовательностях случайных чисел возникают в самых разных областях знаний. Основные из них – моделирование, численный анализ, программирование, теория принятия решений и теория игр. Для всех этих целей вполне пригодным оказывается метод генерации псевдослучайных чисел, который способен выдавать последовательности с внушительной величиной периода и распределением, сколь угодно близким к заданному. Случайные числа находят свое применение и в криптографии. С их помощью происходит генерация ключей в большинстве криптопротоколов, случайных параметров сеансов связи в протоколах аутентификации, значений параметров многих систем ЭЦП и т. д. В этом случае накладывается дополнительное ограничение независимости в совокупности всех получаемых значений случайных чисел, что фактически означает невозможность (или вычислительную неэффективность) получения новых значений случайных чисел по известной части последовательности. Лишь немногочисленный класс генераторов ПСП удовлетворяет этому требованию; как правило, такие генераторы по-прежнему обладают строго детерминированным алгоритмом, однако функция генерации необратима, что затрудняет нахождение зависимостей внутри последовательности. В подобных ситуациях предпочтительнее использовать генератор истинно случайных чисел, реализуемый с помощью некоторого источника внешней энтропии. Под таким источником мы будем понимать некий физический процесс, параметры которого в каждый момент времени очень сложно предсказать. Известно множество реализаций таких аппаратных генераторов, в том числе достаточно остроумных, среди них плата с шумящим диодом, ПЗС-матрица в затемненной камере, датчик радиоактивного фона и многие другие. Одним из недостатков аппаратных генераторов является их относительно высокая стоимость, связанная с покупкой и обслуживанием необходимого оборудования; программные же генераторы

Омский государственный университет им. Ф.М. Достоевского.
E-mail: sbelim@mail.ru

лишены этих неудобств. Одной из базовых задач данного исследования являлся поиск аппаратных средств, входящих в состав большинства типовых конфигураций персональных и рабочих компьютеров по умолчанию и являющихся источником некоторой внешней энтропии. В данном случае стоимость генератора случайных чисел состояла бы исключительно из стоимости ПО для калибровки устройства и считывания значений. В качестве такого оборудования была выбрана звуковая карта, в интегрированном варианте присутствующая на абсолютном большинстве материнских плат. Источником энтропии в данном случае является шум на ее линейном входе. Основной задачей исследования являлось считывание этого шума и изучение его статистических характеристик, позволяющих сделать заключение о пригодности данного генератора для конкретных целей.

1. Реализация генератора случайных последовательностей

В качестве источника внешней энтропии рассматривается линейный вход звуковой карты [1]. Этот разъем предназначен для подключения источников аналогового звукового сигнала (таких как синтезатор или обычный кассетный магнитофон); такой сигнал, принятый на входе звуковой карты, переводится в последовательность байтов с помощью аналого-цифрового преобразователя. К действию полезного сигнала постоянно добавляется шум, вызванный электромагнитными наводками от других элементов цепи, тепловым шумом в цепях питания и прочими флуктуациями в подсистеме аналогового входа. С позиции поиска источника энтропии интерес представляет именно шум, а потому рассматривается линейный вход звуковой карты, на который не поступает никакого полезного сигнала. Считывание данных можно проводить с помощью любого интерфейса взаимодействия с устройством, например силами компонента DirectSound из коллекции DirectX. Так как мощность шума существенно ниже мощности полезного сигнала, предполагается, что в считываемых данных будут задействованы в основном младшие биты, причем максимальная битовая глубина шума напрямую зависит от качества устройства.

Для считывания данных с линейного входа звуковой карты была написана программа SoundRandom. Программа позволяет составить список устройств звукового захвата, действующих на компьютере, и выбрать в качестве источника любое из них. В качестве основных параметров считываемой последовательности можно задавать ее длину, а также количество младших битов, используемых для генерации. Программа позволяет провести базовую оценку равномерности распределения значений с помощью описательной статистики: среднее, сумма, дисперсия, эксцесс и пр. Код программы разделен на два отдельных модуля, отличающихся по функциональному назначению. Модуль взаимодействия с устройством реализует алгоритм считывания данных с линейного входа, используя функции DirectSound, среда разработки - Microsoft Visual C++ 6.0 [2]. Модуль графического интерфейса обеспечивает взаимодействие с пользователем, установку параметров, вывод графиков и отображение результатов, среда

разработки - Borland C++ Builder 6.

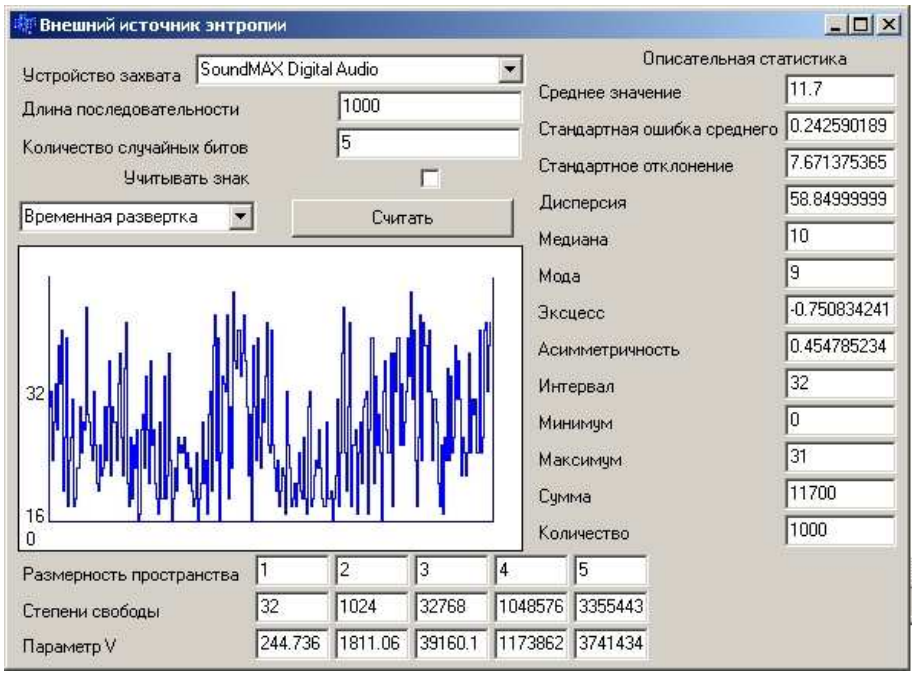


Рис. 1. Интерфейс программы SoundRandom

Алгоритм модуля взаимодействия с устройством состоит из функций, инициализирующих интерфейсы обращения к устройствам захвата и буферам записи, а также из функций, обрабатывающих начало и конец записи и очередное переполнение буфера. Интерфейс считывает по 44100 звуковых сэмплов в секунду, результат передается в вызывающий модуль, где для формирования последовательности используется необходимое количество младших битов сэмпла.

Функции, реализующие функционал модуля:

```
// Определение доступных устройств звукового захвата
HRESULT WINAPI DirectSoundCaptureEnumerate(
    LPDSENUMCALLBACK lpDSEnumCallback,
    LPVOID lpContext
);

// Создание объекта для взаимодействия с устройством
HRESULT WINAPI DirectSoundCaptureCreate(
    LPGUID lpGUID,
    LPDIRECTSOUNDCAPTURE *lplpDSC,
    LPUNKNOWN pUnkOuter
);

// Создание буфера для считывания данных
```

```
HRESULT CreateCaptureBuffer(
    LPDSCBUFFERDESC lpDSCBufferDesc,
    LPLPDIRECTSOUNDCAPTUREBUFFER lplpDirectSoundCaptureBuffer,
    LPUNKNOWN pUnkOuter
);

// Разметка буфера для определения переполнения
HRESULT SetNotificationPositions(
    DWORD cPositionNotifies,
    LPCDSBPOSITIONNOTIFY lpcPositionNotifies
);
```

2. Тестирование случайных последовательностей

Опираясь на данные одной лишь описательной статистики, нельзя утверждать, что полученная последовательность является равномерно распределенной случайной последовательностью (РРСП). Поэтому встает вопрос о проверке результатов с помощью ряда статистических критериев. Статистический критерий - это некая процедура, применяемая к количественным данным выборки. Каждый тест предназначен для проверки гипотезы H: «Заданная последовательность имеет равномерно распределенную случайную структуру». Для этого рассчитывается определенная статистика (своя для каждого теста), которая сравнивается с соответствующей статистикой РРСП. Величина отклонения полученного параметра от эталонного служит мерой «подозрительности» результата; начиная с некоторого предела отклонения последовательности отвергаются как неслучайные (или же распределенные иначе). Прохождение одного теста не гарантирует совпадения всех параметров последовательности с параметрами РРСП, поэтому очень важно прохождение целого набора тестов. Количество уже разработанных критериев велико, в сущности, их можно изобрести сколь угодно много. Хорошо зарекомендовавшими себя являются алгоритмы, описанные в «Искусстве программирования» Д. Кнута [1], а также тесты из большого набора «DieHard Test Battery» Д. Марсалья [4]. Опробовав полученные последовательности на достаточном количестве критериев, можно сделать вывод о ее пригодности к применению в криптостойких алгоритмах.

Для анализа статистических свойств сгенерированных числовых последовательностей создана программа Analizer. Она содержит набор из шести статистических критериев и механизм для обработки с их помощью последовательностей при любом заданном уровне группировки элементов [5,6]. По расчетным критериям программа представляет результат о прохождении данного теста и о категории, в которую попала тестируемая последовательность.

Критерии, реализованные в приложении: критерий хи-квадрат, критерий Колмогорова-Смирнова, критерий интервалов, покер-критерий, критерии распределения на плоскости и в пространстве.

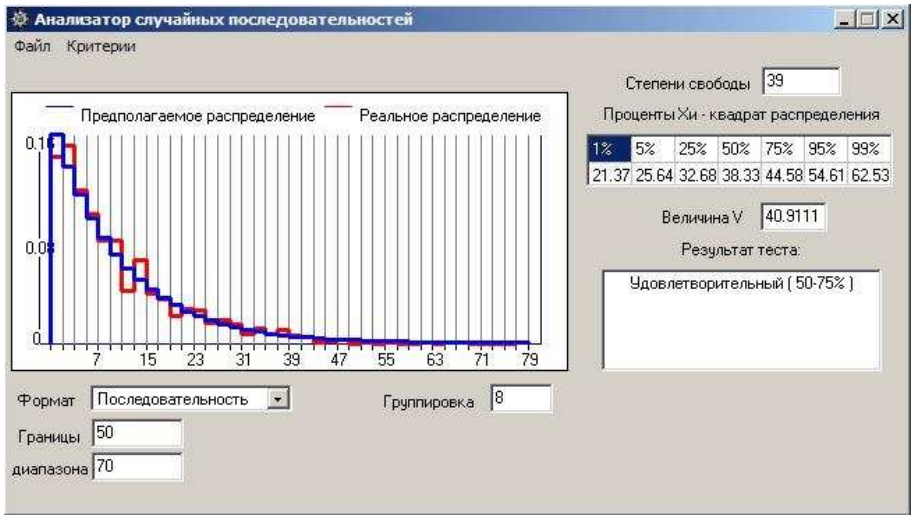


Рис. 2. Интерфейс программы SoundRandom

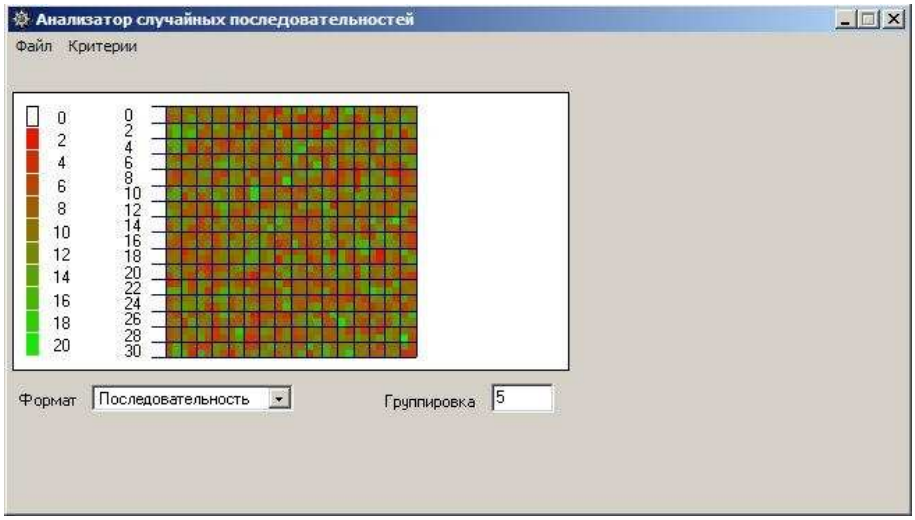


Рис. 3. Интерфейс программы SoundRandom

Испытания проводились на нескольких типах звуковых карт, все экземпляры были в интегрированном исполнении. Отсутствие сигнала на линейном входе карты контролировалось. С каждого устройства было считано по три последовательности длиной 100000 бит. Результаты применения тестов приведены в таблице; критерии, выдавшие неоднозначный ответ, отмечены цветом. Как видно, большинство тестов проходятся благополучно.

Картам в таблице присвоены следующие номера: 1) Realtek ALC662, 2) CMI8738/C3DX PCI, 3) SoundMax Digital Audio, 4) VIA AC97 Audio Chipset, 5) ADI AD1986A HD Audio. Набор тестов: T1 - хи-квадрат, T2 - критерий Колмогорова-Смирнова, T3 - критерий интервалов, T4 - покер-критерий. Параметры группировки: (1) - по 1 биту, (2) - по 2 бита, (3) - по 3 бита, (4) - по 4 бита.

Таблица 1. Результаты проверки критериев

посл	T1(1)	T2(1)		T3(1)	T4(1)	T1(2)	T2(2)		T3(2)	T4(2)
1 - 1	4.17	0	1.02	13.34	0.48	6.79	0	0.76	36.35	5.3
1 - 2	2.48	0.79	0	24.64	0.14	2.85	0.54	0	33.25	7.79
1 - 3	0	0.03	0	19.8	0.65	0.12	0.08	0.02	39.19	6.18
2 - 1	0	0	0.01	64.46	2.07	0.49	0.04	0.16	35.87	6.99
2 - 2	1.35	0	0.58	18.29	5.13	5.04	0	0.68	82.71	1.94
2 - 3	1.49	0	0.61	16.36	0.42	2.47	0	0.48	40.23	8.1
3 - 1	2.12	0	0.73	9.44	2.14	3.95	0	0.57	18.56	4.77
3 - 2	2.06	0.72	0	27.16	0.47	4.13	0.58	0	24.95	2.7
3 - 3	0.92	0.48	0	15.77	1.49	1.1	0.29	0	74.64	2.3
4 - 1	1.78	0	0.67	39.81	7.96	8.02	0.28	0.39	70.75	13.24
4 - 2	2.72	0	0.85	19.17	4.25	7.25	0	0.79	16.89	4.31
4 - 3	1.63	0	0.64	15.43	2.92	3.72	0	0.57	60.39	0.4
5 - 1	0.16	0	0.2	16.2	0.21	0.63	0.02	0.22	11.24	6.14
5 - 2	0.81	0	0.45	19.01	10.01	10.1	0.3	0.75	65.13	5.99
5 - 3	0.1	0.16	0	2.4	5.41	1.17	0.28	0.03	23.95	2.85

посл	T1(3)	T2(3)		T3(3)	T4(3)	T1(4)	T2(4)		T4(4)
1 - 1	8.48	0	0.58	112.87	9.66	16.08	0	0.5	17.62
1 - 2	5.09	0.4	0	62.23	5.68	24.95	0.47	0.19	4.38
1 - 3	4.04	0.1	0.35	34	6.85	19.57	0.26	0.14	4.31
2 - 1	11.92	0.18	0.52	67.17	1.63	20.37	0.32	0.25	1.87
2 - 2	6.14	0.04	0.48	31.78	2.99	19.09	0.28	0.44	8.11
2 - 3	4.82	0.05	0.33	26.13	7.05	14.21	0	0.47	5.07
3 - 1	14.94	0.24	0.82	158.96	8.3	23.15	0.24	0.39	3.12
3 - 2	8.2	0.3	0.14	32.7	3.6	12.4	0.33	0.1	2.06
3 - 3	14.94	0.48	0.23	35.22	3.53	16.07	0.32	0.08	6.58
4 - 1	13.76	0.14	0.54	251.39	10.57	30.49	0.25	0.53	3.35
4 - 2	13.55	0.07	0.64	75.7	7.28	22.51	0.22	0.34	2.56
4 - 3	5.2	0	0.37	50.83	1.79	19.02	0.13	0.45	1.71
5 - 1	4.97	0.26	0.18	64.59	6.57	16.22	0.16	0.27	3.87
5 - 2	13.36	0.37	0.38	66.44	5.25	43.13	0.45	0.37	2
5 - 3	3.74	0.24	0.11	40.26	1.43	15.98	0.43	0.15	5.31

В результате проведенных исследований был сделан вывод о том, что аналоговый вход звуковой карты обладает большим потенциалом в плане конструирования генераторов «истинно» случайных чисел. Основной положительный фактор в данном вопросе – цена построения работающего устройства; пользователь платит лишь за ПО, в аппаратной же части ему не нужно докупать никаких дополнительных приспособлений. В то же время генерируемые последовательности выдерживают все тесты на случайность и равномерность распределения, а потому годятся для применения в областях с повышенными требованиями к случайным числам. Также данный способ отличается большой стойкостью к возможным компрометациям со стороны конкурентов и недоброжелателей в случае, если будет предпринята попытка доказать прогнозируемость получаемой последовательности. Скомпрометировать генератор возможно только на специальным образом сконфигурированной системе, либо явно вмешавшись в его работу. С другой стороны, соблюдая определенные требования (спецификацию), можно быть уверенными в достаточной непредсказуемости генерируемых случайных чисел.

Литература

1. Юрьев Л. Генерация истинно случайных чисел на основе шума звуковых карт. URL: http://leo.yuriev.ru/114 (дата обращения: 01.02.2010).
2. Горнаков С.Г. DirectX 9: Уроки программирования на C++. Спб.: БХВ-Петербург, 2005. 400 с.
3. Кнут Д. Э. Искусство программирования. Том 2. Случайные числа. М.: Вильямс, 2002. 720 с.
4. Ylonen T. Introduction to Cryptography.
URL: http://algolist.manual.ru/defence/intro.php (дата обращения: 10.02.2010).
5. Иванов М.А., Чугунков И.В. Теория, применение и оценка качества генераторов псевдослучайных последовательностей. М: КУДИЦ-ОБРАЗ, 2003. 240 с.
6. Marsaglia G. DIEHARD Statistical Tests. URL: http://stat.fsu.edu/ geo/diehard.html (дата обращения: 01.02.2010).

Математические структуры и моделирование
2010, вып. 21, с. 44–54

УДК 004.4'2

ВЫСОКОУРОВНЕВАЯ МОДЕЛЬ СЕМЕЙСТВА ПРОГРАММНЫХ КОМПОНЕНТОВ ДЛЯ ПОДДЕРЖКИ ЗАНЯТИЙ В ИГРОВОЙ ФОРМЕ

С.В. Гусс

В работе представлена классификация элементов повторного использования, указано место в ней каркаса как основы для разработки семейства программных продуктов. Даётся краткое описание его отличия от других представителей классификации. Приводится описание высокоуровневой модели разработанного автором каркаса в виде диаграмм языка UML и текстовых пояснений.

Введение

С тех пор как компьютерные программы стали рассматриваться разработчиками как инженерные творения и создавались уже не единичные программы, решающие конкретную проблему, а продукты, принадлежащие определённому программному семейству, проблема повторного использования приобрела большое значение. О возможности создания семейства программных продуктов, в отличие от единичных программ, говорил в одной из своих статей Давид Парнас [1].

На протяжении многих лет занятия в игровой форме, например лингвистические, давали результат в обучении не только языку и его особенностям, но и применению его в профессиональной сфере. Об этих занятиях можно говорить так же, как и о других обучающих занятиях в игровой форме. Большая работа в этом направлении проделана Марком Пренским, исследователем в области электронного обучения, в основе которого лежат компьютерные игры. В одной из его публикаций [2] можно найти информацию о том, почему именно игровая форма является привлекательной. Ниже представлен список причин, взятый из книги.

1. Игры доставляют удовольствие.
2. Способствуют интенсивному вовлечению в процесс обучения.
3. В них заложены правила, из чего следует упорядоченность и структура процесса.

Омский государственный университет им. Ф.М. Достоевского.
E-mail: InfoGuss@gmail.com

4. Они мотивируют, так как имеют определённые цели.
5. Они интерактивны, что заставляет действовать.
6. У них обязательно есть обратная связь, что способствует обучению.
7. Для игрового процесса характерно свойство привыкания, что позволяет без труда получать новые знания посредством игры.
8. Победа, одержанная в игре, даёт возможность почувствовать свои силы.
9. Они развивают творческие способности благодаря тому, что заставляют справляться с проблемами, искать способы их решения.
10. Игры способны развить коммуникационные способности благодаря специальным многопользовательским режимам.
11. Могут воздействовать на эмоции.

В этой же публикации есть описание возможных игровых стилей для различных тематических направлений изучаемых дисциплин.

1. Место каркаса в классификации элементов повторного использования

Часто в литературе вместо фразы «элементы повторного использования» можно встретить словосочетание «компоненты многократного применения». Они не являются синонимами, но это описание одного и того же явления на разных уровнях представления. В первом случае речь идёт об абстрактном уровне представления, в последнем - о конкретном, или уровне реализации, когда имеют в виду наличие готового к использованию компонента, а не описывающую его модель. Элементы повторного использования можно классифицировать по нескольким признакам.

1. По уровню представления:
   - Модели. Представляют собою высокоуровневое описание структуры и поведения.
   - Набор кода. Готовый к применению код на определённом языке программирования.
   - Исполняемые модули. Компоненты, готовые к запуску пользователем или другим программным приложением. Характеризуются отсутствием привязки к языку программирования. Но возможна привязка к определённой программно-аппаратной платформе.
2. По специфике повторно используемого элемента. Классификация похожа на ту, что приведена в [3]. Только в представленной ниже классификации не фигурирует такой элемент, как модель, т.к. она не подходит под данный признак деления, в связи с чем этот элемент был отнесён к другой группе.
   - Функциональные библиотеки (набор подпрограмм под специфические нужды). В отличие от каркасов здесь повторно используются функции, а написание тела приложения, в котором будут использованы эти функции, остаётся задачей разработчика.
   - Образцы (соответствуют этапам разработки):
     – анализа [4] (применимы для анализа предметной области прило-

жения);
  - архитектурные [5] (для упорядочивания архитектуры и реализации определённых характеристик качества);
  - проектирования [6] (для применения опыта, накопленного и обобщённого другими разработчиками в виде особых проектных решений);
  - реализации [7] (для обслуживания задач ежедневного программирования, а также для придания программному коду читабельного вида и внесения в него большей ясности).
- Каркасы. Более специализированы, чем образцы. Важно отметить, что приложение может разрабатываться на основе нескольких каркасов; это довольно часто относится к крупным приложениям масштаба предприятия. Особенность состоит в том, что повторно используется тело приложения, к которому добавляются функции и уточнения.

3. По этапу разработки (элементы могут быть представлены и моделями, и набором кода, и в виде исполняемых модулей, и даже просто в виде текстового описания):
   - Предметной области.
   - Сбора и анализа требований.
   - Архитектуры. Чтобы архитектура могла быть многократно используемой, крайне необходимо отделить её от реализации, а на основании [8] - ещё и от алгоритмов и представления данных.
   - Проекта.
   - Реализации.
   - Тестирования.

2. Проблема разработки каркаса

Каркас – это «набор взаимодействующих классов, составляющих повторно используемый дизайн для конкретного класса программ» [6]. Конкретный класс программ, или семейство программных продуктов – набор программ, имеющих так много общего, что выгоднее изучить их общие аспекты, прежде чем изучать те аспекты, в которых они различны [1]. Влияние каркаса на детализирующую его систему может быть описано следующими принуждающими параметрами [6]:

1. Определённая архитектура.
2. Структура классов и объектов.
3. Основная функциональность, описанная в классах.
4. Методы взаимодействия между объектами.
5. Потоки управления.

Необходимость этих параметров обусловлена тем, что разработчики могут сконцентрироваться на специфике разрабатываемой программы и не тратить время на поиск черт, уже описанных и известных. Одна из проблем дизайна любого типа программного обеспечения заключается в том, что, как отмечается в [6], дизайн должен соответствовать задаче, но в то же время быть общим для того,

чтобы была возможность учесть требования, возникновение которых возможно в будущем. Сложность состоит ещё и в том, чтобы разложить систему на объекты. Кроме того, «вся система должна обладать концептуальным единством» [9].

3. Описание модели каркаса

Концепция. Главными пользователями каркаса являются разработчики программного обеспечения. Реализация модели каркаса, описываемой в данной работе, позволяет ускорить процесс разработки компонентов программного продукта за счёт того, что в нём уже заложена стратегия управления всеми необходимыми операциями, действия которых происходят в игровом процессе. Разработчику, использующему каркас, необходимо лишь уточнить в нём некоторые детали и заложить отдельные стратегии реализации операций, специфичных для решаемой задачи. Пользователем разработанного на основе каркаса компонента также является разработчик. Пользователем продукта, построенного на основе этого компонента или компонентов, будет уже не разработчик, а конечный пользователь, для которого создавался продукт.

Взаимодействие с пользователем. Назовём внешним субъектом каркаса в его детализированном состоянии (то есть в состоянии, когда каркас подведён под нужды соответствующей задачи) объект типа «Клиент», представляющий собой клиента системы, реализуемой посредством компонента. Объект типа «Клиент» обращается к подсистеме главным образом через вызов операций «Начать игру» и «Сделать ход».

Диаграмма классов каркаса представлена на рисунке 2.

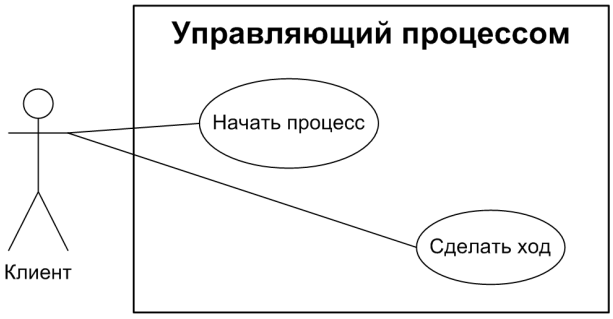


Рис. 1. Диаграмма вариантов использования детализированного каркаса

Структуру каркаса условно можно разделить на два уровня:

1. Уровень основных элементов.
2. Уровень управления процессом.

Элементы уровня основных элементов:

1. «Игрок». Представляет собой главного участника процесса, роль которого может исполнять посредством системы объект типа «Клиент». «Игрок» связан с объектами типов «Соперник» и «Судья».
2. «Соперник». Роль, исполняемая вычислительной машиной, представляет собой противника для объекта типа «Игрок».

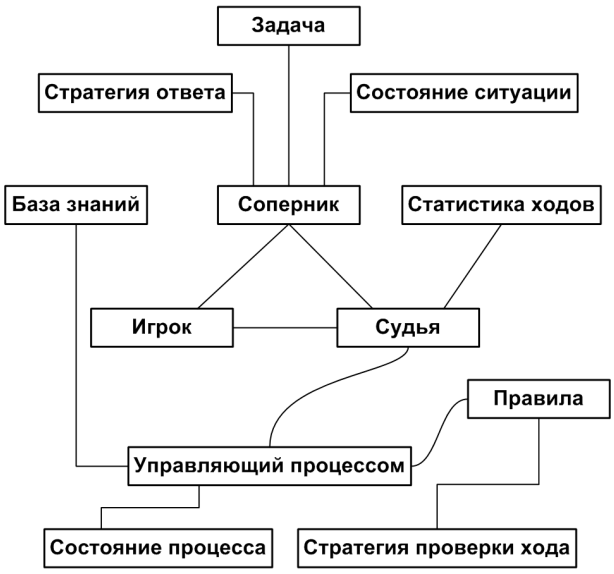


Рис. 2. Диаграмма классов каркаса

3. «Судья». Роль, исполняемая вычислительной машиной, является арбитром игрового процесса.
4. «Задача». Тип, представляющий собой задачу, решаемую во время занятий в игровой форме.
5. «Стратегия ответа». Должна быть задана во время детализации каркаса. В проекте системы может быть реализована посредством шаблона «Стратегия» [6].
6. «Статистика ходов». Тип статистики событий процесса.
7. «Состояние ситуации». Показывает состояние решения задачи. Возможные варианты решения задачи:
   - «Ещё не решена» - означает, что задача ещё не решена.
   - «Почти решена» - некоторые части задачи еще не решены.
   - «Решена» - задача решена полностью.

Элементы уровня управления процессом:

1. «Управляющий процессом». Тип объекта, представляющего сущность, знакомую с элементами процесса и принципами его организации. Содержит в себе описание события «Обработка вопроса», вызываемого действиями объекта типа «Игрок», когда он обменивается сообщениями с объектом типа «Соперник» во время игры. На это событие объект типа «Клиент» должен подписаться до начала процесса.
2. «Правила». Описание правил процесса.
3. «Стратегия проверки хода». Так же, как и стратегия ответа, должна быть задана во время детализации каркаса.
4. «База знаний». Тип, описывающий объекты, способные управлять знаниями.
5. «Состояние процесса». Описывает состояние процесса. Процесс может находиться в двух состояниях:

- «Активный» - процесс в активном состоянии.
- «Неактивный» - процесс находится в неактивном состоянии.

Поток управления. Детализация (или специфицирование под конкретный случай) каркаса происходит следующим образом. Создаётся новый тип (класс в терминах объектно-ориентированной парадигмы), производится операция наследования этим типом структуры и поведения, присущих классу «Управляющий процессом». Задаётся стратегия проверки хода, сделанного игроком, и стратегия ответа или, другими словами, реакция на ход игрока. Помимо этого необходимо создать задачу и присвоить её представление объекту типа «Задача», доступного благодаря наследованию от класса «Управляющий процессом». Диаграмма последовательностей детализации каркаса представлена на рисунке 3.

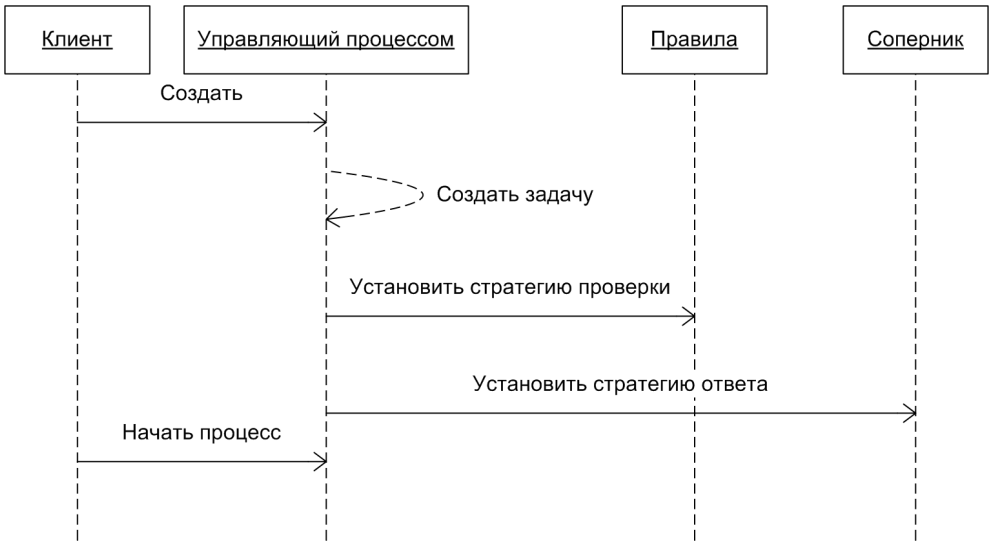


Рис. 3. Диаграмма последовательностей детализации игрового каркаса

Последовательность взаимодействия с детализированным каркасом (рисунки 4 и 5) предполагает следующие действия:

1. Создание экземпляра класса (обязанность объекта типа «Клиент»), наследованного от «Управляющий процессом», который создаёт экземпляры следующих типов:
   - «Игрок»,
   - «Соперник»,
   - «Судья»,
   - «Правила»,
   - «База знаний».
2. Старт процесса.
3. Осуществление игрового хода. После чего происходит событие обработки вопроса, передающее объекту типа «Клиент» информацию о проделанном игровом ходе.

Описание поведения. При выполнении операции «Сделать ход» происходит следующая последовательность действий:

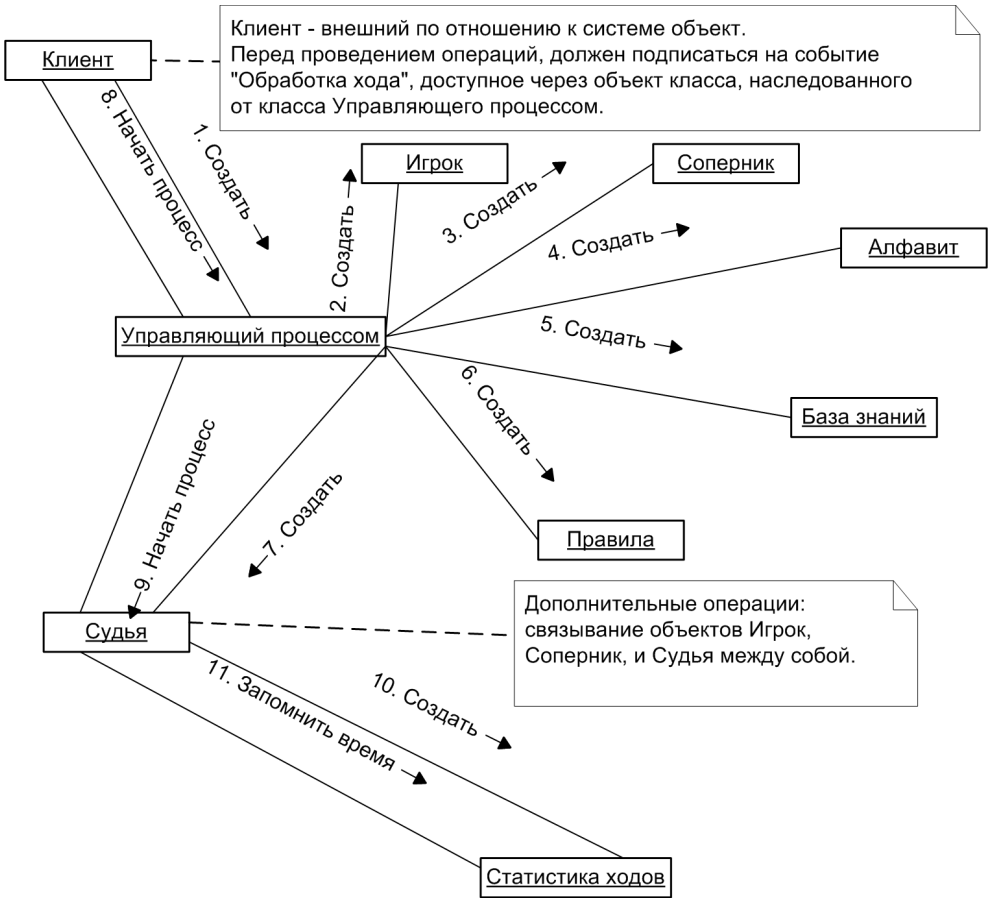


Рис. 4. Диаграмма сотрудничества объектов каркаса во время операции «Начать процесс»

1. После получения в качестве входного параметра списка вопросов осуществляется проверка условия «Процесс находится в активном состоянии». При положительном ответе на утверждение условия происходит переход на следующий шаг, иначе – выход из подпрограммы.
2. Передаётся сообщение объекту типа «Игрок» - «Совершить ход».
3. Происходит проверка состояния задачи, после чего, если состояние изменилось, происходит фиксация этого изменения.
4. Происходит событие обработки вопроса, заданного объектом типа «Игрок» во время хода.

При выполнении объектом типа «Игрок» операции «Совершить ход» происходит следующая последовательность действий:

1. После получения в качестве входных параметров списка вопросов и объекта типа «Правила» производится операция - «Получить ответ от соперника».
2. Обновляется состояние решения задачи.
3. Происходит обновление статистики в объекте типа «Судья».
4. Возвращается состояние задачи.

При выполнении операции «Получить ответ от соперника» происходит сле-

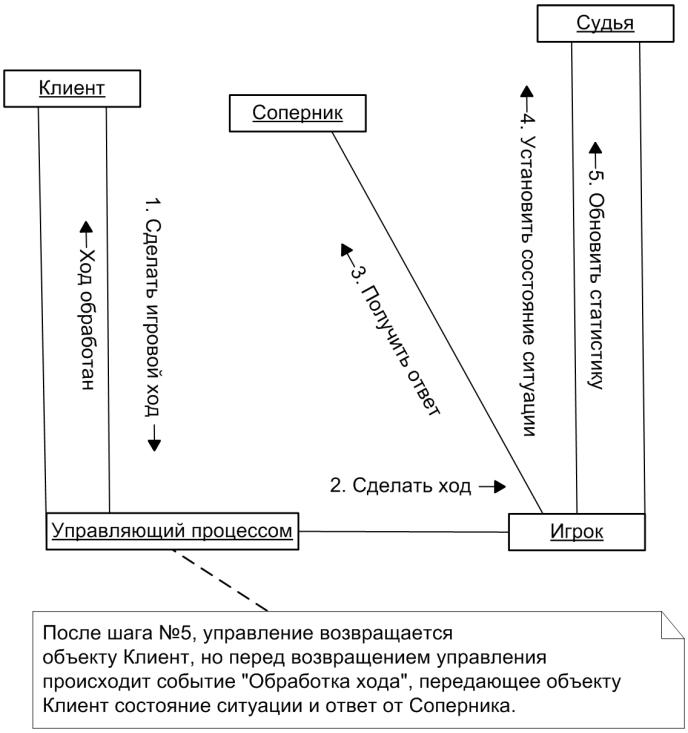


Рис. 5. Диаграмма сотрудничества объектов каркаса во время операции «Сделать игровой ход»

дующая последовательность действий:

1. После получения в качестве входных параметров списка вопросов и правил игры выполняется операция проверки хода в соответствии с правилами.
2. Происходит проверка результата предыдущего действия. При удачной проверке происходит выполнение операции «Получить ответ» согласно заданной стратегии, описанной тем, кто производил детализацию каркаса. В противном случае система вынуждена сообщить о некорректности хода.

При выполнении операции «Получить результат проверки» происходит последовательность действий, приведённых ниже:

1. После получения в качестве входных параметров списка вопросов и объекта типа «Судья» происходит проверка условия «Вопрос задан в первый раз». Если условие выполняется, происходит переход на шаг 2, иначе – на шаг 4.
2. Происходит получение списка вопросов, заданных в процессе игры.
3. Проверяется наличие текущего вопроса в списке. При условии нахождения запрашиваемого элемента возвращается результат «Имело место повторение».
4. Происходит выполнение операции «Проверить вопросы» в соответствии с заданной стратегией, описанной тем, кто производил детализацию каркаса.

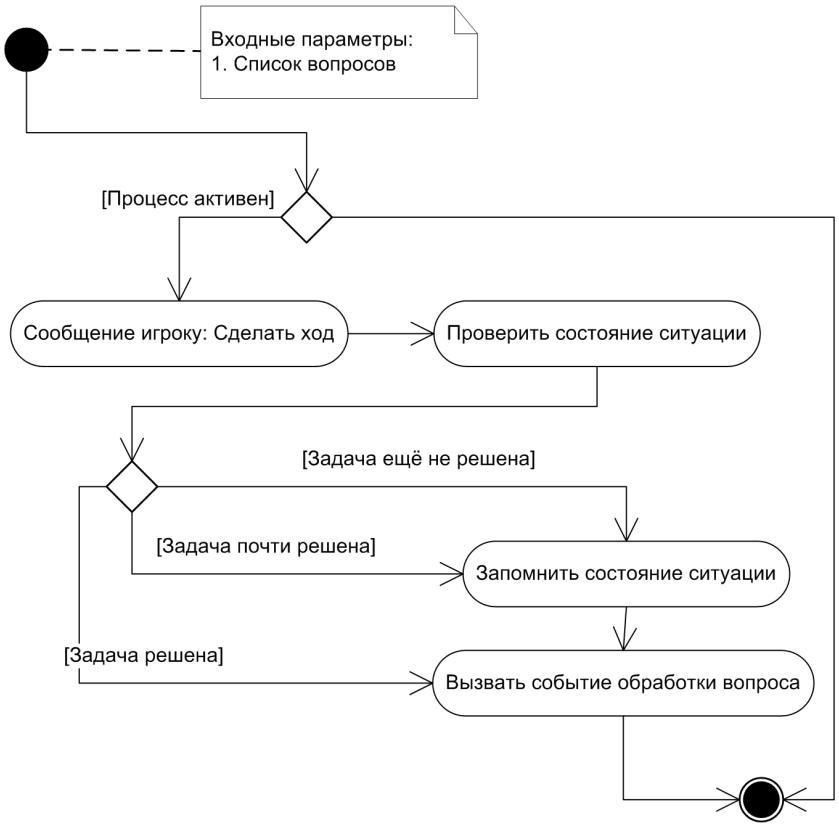


Рис. 6. Диаграмма деятельности, описывающая последовательность действий при вызове операции «Сделать ход»

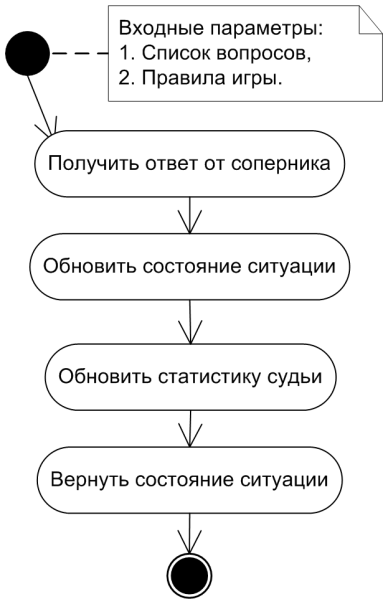


Рис. 7. Диаграмма деятельности, описывающая последовательность операций при выполнении операции «Совершить ход» на объекте типа «Игрок»

Рис. 8. Диаграмма деятельности, описывающая последовательность действий при вызове операции «Получить ответ от соперника»

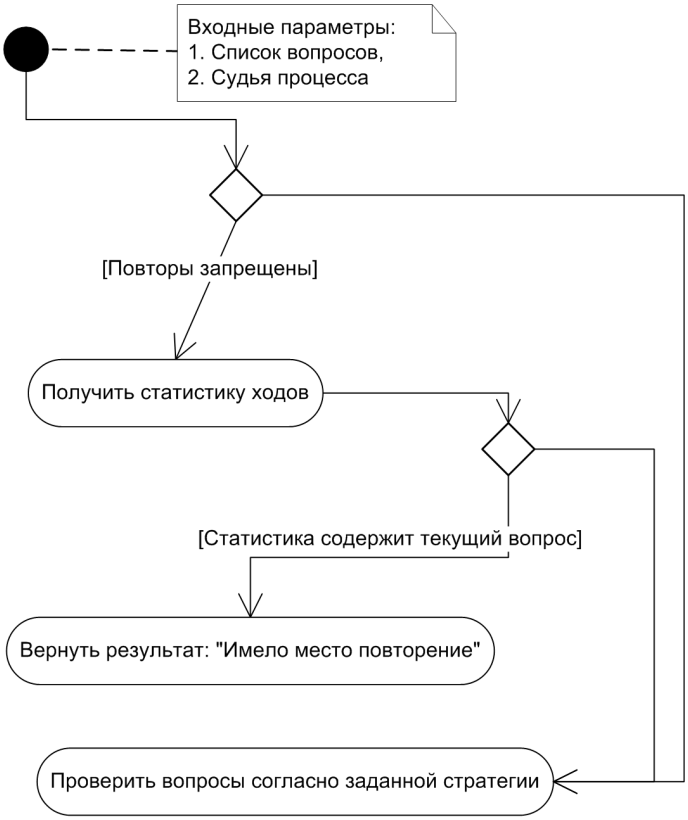


Рис. 9. Диаграмма деятельности, описывающая последовательность операций при вызове метода «Получить результат проверки»

Заключение

В работе была представлена высокоуровневая модель каркаса. Реализация описанной модели способствует упрощению процесса создания программных компонентов поддержки занятий в игровой форме. Модель использовалась автором для создания проекта игровых компонентов. Воплощение этого проекта способствовало реализации готовых компонентов, которые были применены в процессе разработки одного из обучающих игровых приложений.

Литература

1. Parnas, D.L. On the Design and Development of Program Families // IEEE Transactions on Software Engineering, Vol. SE-2, No. 1, March 1976. P. 1–9.
2. Prensky M. Digital Game-Based Learning. U.S.A.: McGraw-Hill, 2004.
3. Рамбо Д., Блаха М. UML 2.0: объектно-ориентированное моделирование и разработка, 2-е издание. СПб.: Питер, 2007.
4. Fowler M. Analysis Patterns: Reusable Object Models. U.S.A.: Addison-Wesley, 1997.
5. Fowler M. Patterns of Enterprise Application Architecture. U.S.A.: Addison-Wesley, 2002.
6. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2008.
7. Бек К. Шаблоны реализации корпоративных приложений. М.: ИД Вильямс, 2008.
8. Басс Л., Клементс П., Кацман Р. Архитектура программного обеспечения на практике, 2-е издание. СПб.: Питер, 2006.
9. Брукс Ф. Мифический человеко-месяц или как создаются программные системы. СПб.: Символ-Плюс, 2001.

Математические структуры и моделирование
2010, вып. 21, с. 55–64

УДК 519.68:530.15

АРХИТЕКТУРА, ПРОЦЕССОР И РАБОТА КВАНТОВОГО КОМПЬЮТЕРА

А.К. Гуц

Краткое изложение устройства квантового компьютера и организации вычислений на нем.

Теория квантового компьютера основывается на *квантовой механике*, созданной в 20-е годы XXI века австрийцем Эрвином Шрёдингером и немцем Вернером Гейзенбергом.

Идея использовать квантовую механику для построения квантового компьютера независимо высказана россиянином Юрием Маниным [4] и американцем Ричардом Фейнманом [5, 6].

1. Архитектура квантового компьютера

Квантовый компьютер имеет архитектуру аналогичную классическому компьютеру. Он состоит из:

– регистров памяти,

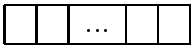
– процессора, построенного из логических элементов и производящего вычисления,

– устройства ввода информации,

– устройства вывода полученной в ходе вычислений информации.

2. Регистры

Память компьютера разбита на регистры. Регистры состоят из некоторого количества разрядов. Регистр из m разрядов изобразим как



Квадратик изображает разряд

Омский государственный университет им. Ф.М. Достоевского.
E-mail: aguts@mail.ru

2.1. Бит и классический регистр

Классический разряд □ хранит единицу информации – *бит* информации – 0 или 1.

Запишем разряд в символическом виде

$$|n_k\rangle, \quad n_k = 0, 1.$$

Тогда классический регистр можно представить как

$$|n_{m-1}n_{m-2}...n_0\rangle. \tag{1}$$

Для технической реализации бита используются разные физические устройства.

Пример 1. Высокий потенциал в точке схемы – 1, низкий – 0.

Пример 2. Ферромагнитное колечко намагничено в одном направлении – 1, в другом – 0.

Состояние классического регистра в момент времени t

Разряд классического регистра находится только в одном из двух возможных состояний – $|0\rangle$ или $|1\rangle$.

Поэтому состояние регистра – это

$$|n_{m-1}n_{m-2}...n_0\rangle.$$

Например, состояние

$$|\underbrace{01001011100011...}_{m}\rangle.$$

2.2. Кубит и квантовый регистр

Квантовый разряд □ хранит единицу информации – квантовый бит, или *кубит* информации – 0 или 1.

Квантовый разряд в символическом виде выглядит так же, как классический:

$$|n_k\rangle, \quad n_k = 0, 1.$$

И поэтому квантовый регистр представляется в виде

$$|n_{m-1}n_{m-2}...n_0\rangle. \tag{2}$$

Для технической реализации кубита предлагаются разные физические устройства, основой которых является любая двухуровневая (квантовомеханическая) система (спин, фотон, атом, молекула, ион).

Пример 1. Проекция спина атома $(+1)$ принимается для кубита за состояние $|0\rangle$, а проекция (-1) – за состояние $|1\rangle$.

Пример 2. Берётся раствор молекул и помещается при комнатной температуре во внешнее магнитное поле. При этом атомные ядра, обладающие ядерным спином, т.е. являющиеся как бы маленькими магнитами, займут одно из двух положений – по полю – это $|0\rangle$ и против него – это $|1\rangle$.

Состояние квантового регистра в момент времени t

Разряд квантового регистра находится в состоянии

$$\alpha|0\rangle + \beta|1\rangle,$$

$$\alpha, \beta \in \mathbb{C}.$$

Поэтому состояние квантового m-разрядного регистра – это *когерентная суперпозиция* всех *базисных состояний*:

$$|\psi(t)\rangle \equiv \sum_{n_{m-1}=0}^{1} \sum_{n_{m-2}=0}^{1} ... \sum_{n_0=0}^{1} c_{n_{m-1}n_{m-2}...n_0}|n_{m-1}n_{m-2}...n_0\rangle, \tag{3}$$

$$c_{n_{m-1}n_{m-2}...n_0} \in \mathbb{C}.$$

Числа $|c_{n_{m-1}n_{m-2}...n_0}|^2$ интерпретируются как вероятность пребывания регистра в состоянии $|n_{m-1}n_{m-2}...n_0\rangle$.

Например, состояние

$$|\psi(t)\rangle \equiv c_1|\underbrace{01001011100011...}_{m}\rangle +$$

$$+c_2|\underbrace{11001011100011...}_{m}\rangle + ...$$

Комментарий. Состояние 1-разрядного регистра квантового компьютера

в момент времени t подобно одновременному пребыванию кота в живом и мёртвом состоянии:

Рис. 1. Живо-мёртвое состояние кота

Иначе говоря, в квантовом мире альтернативы могут существовать одновременно.

Если предположить, что квантовый компьютер находится сразу во множестве параллельных вселенных, то в одной вселенной кот жив, а в другой мёртв. Наблюдатель видит только того кота, в какой вселенной живет сам; параллельный, другой мир он не видит. Такой подход называется эвереттовской интерпретацией квантовой механики [2].

3. Процессор

Процессор компьютера служит для того, чтобы менять состояние регистров.

Делается это посредством физического воздействия на биты в классическом компьютере и на кубиты – в квантовом; в результате и те и другие меняют свое состояние.

> **Пример**. Если кубит представляет собой атом, то регистр – это квантовая система из m атомов. Воздействие на эту систему осуществляется с помощью специально подобранных импульсов лазеров. Лазерные импульсы влияют на электронные состояния атомов.
>
> Лазерными импульсами управляет уже классический компьютер, входящий в состав того устройства, которое мы назвали квантовым компьютером.

3.1. Классический процессор

Процессор классического компьютера состоит из схем, собранных из логических элементов.

Логический элемент – это простейшее устройство ЭВМ, выполняющее одну определенную логическую операцию над входными сигналами согласно правилам алгебры логики.

Для логических элементов независимо от их физической реализации приняты дискретные значения входных и выходных сигналов; обычно это два уровня, которые условно принимаются за 0 и 1.

На рис. 2 изображен логический элемент «НЕ», который переводит 0 в 1 и 1 в 0.

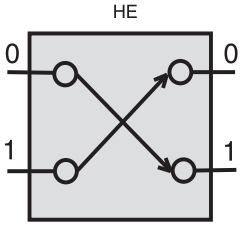


Рис. 2. Логический элемент «НЕ»

Логические элементы – это технические устройства, реализующие некоторые логические операции классической логики. Так, элемент «НЕ» соответствует операции $\neg$, элемент «ИЛИ» – операции $\vee$ и т.д. Установлено, однако, что элементная база классического компьютера основывается всего на двух логических элементах, например на «НЕ» и «исключающее ИЛИ-НЕ».

Процессор преобразует, меняет содержание разрядов регистра посредством каждого входящего в него логического элемента U:

$$U : |n_{m-1}n_{m-2}...n_0\rangle \to |n'_{m-1}n'_{m-2}...n'_0\rangle. \qquad (4)$$

3.2. Квантовый процессор

Квантовый процессор также состоит из логических элементов, называемых *гейтами.*

На рис. 3 изображен квантовый логический элемент «$\sqrt{\text{НЕ}}$», который переводит 0 в 1 и 1 в 0, а также 0 в 0 и 1 в 1, но лишь с вероятностью $p_{ij} = 1/2,\ (i, j = 0, 1)$.

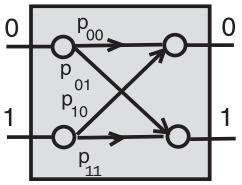


Рис. 3. Логический элемент с **вероятностями** p_{ij} переходов $i \to j$.

В теории квантового компьютера существует бесконечное количество логических элементов. Однако доказано, что квантовый компьютер может быть построен всего из двух логических элементов: однокубитового $\widehat{Q}(\theta, \varphi)$ и 2-кубитового «$\widehat{CNOT}$» (управляемое «НЕ»).

Квантовый процессор преобразует, меняет содержание разрядов квантового регистра посредством каждого входящего в него логического элемента (гейта) $\widehat{U}$:

$$\widehat{U} : \sum_{n_{m-1}=0}^{1} \sum_{n_{m-2}=0}^{1} ... \sum_{n_0=0}^{1} c_{n_{m-1}n_{m-2}...n_0} |n_{m-1}n_{m-2}...n_0\rangle \to$$

$$\rightarrow \sum_{n_{m-1}=0}^{1} \sum_{n_{m-2}=0}^{1} ... \sum_{n_0=0}^{1} c_{n_{m-1}n_{m-2}...n_0} |n'_{m-1}n'_{m-2}...n'_0\rangle. \quad (5)$$

Как видно из формулы (5), **в один шаг изменены сразу все 2^m значений базисных состояний**. Это *эффект параллелизма* в работе квантового компьютера, не имеющий места для классических компьютеров. Для такой производительности за один шаг потребовалось бы 2^m классических процессоров. Если $m = 16$, то $2^{16} = 65536$!

Состояние (3) называется *сцепленным*, если оно не может быть представлено в виде

$$|x_1\rangle...|x_m\rangle, \quad (6)$$

где $|x_j\rangle = \alpha_j|0\rangle + \beta_j|1\rangle$ $(j = 1, ..., m)$.

Если состояние регистра (3) не является сцепленным, то это означает фактическое наличие в распоряжении для вычислений классического регистра вида (6), а значит, только он и преобразуется на данном такте работы квантового компьютера. Отсутствуют другие, параллельные базовые состояния, и, следовательно, отсутствует эффект квантового параллелизма, существенно ускоряющего работу компьютера и определяющего беспрецедентную эффективность квантовых вычислений.

4. Условия для того, чтобы появился квантовый компьютер

Для того чтобы квантовый компьютер стал реальным инструментом для вычислений, необходимо решить следующие технические проблемы:

- Создать физическое устройство, содержащее достаточно большое число N $>$ 100 кубитов;
- Научиться приводить входной регистр в исходное основное базисное состояние
$$|\underbrace{00...0}_{m}\rangle;$$
- Обеспечивать большое время декогеренции (не менее, чем в 104 раза больше времени выполнения основных квантовых операций (время такта)).

  Декогеренция – это взаимодействие системы кубитов с окружающей средой. Она приводит к разрушению суперпозиций квантовых состояний и делает невозможным выполнение квантовых алгоритмов.

- Обеспечить возможность измерения состояния квантовой системы на выходе, то есть при выводе результата.

Рис. 4. Устройство квантового компьютера [1].

5. Вычисление на квантовом компьютере

5.1. Ввод начальных данных

Дано базовое состояние регистра (памяти):

$$|\underbrace{00...0}_{m}\rangle \equiv \underbrace{|0\rangle \otimes |0\rangle \otimes ... \otimes |0\rangle}_{m}. \tag{7}$$

С помощью последовательного применения к состоянию (7) гейтов

$$\widehat{U}^{(1)}, \widehat{U}^{(2)}, ..., \widehat{U}^{(m)},$$

$$\widehat{U}^{(k)} = \widehat{I} \otimes ... \otimes \widehat{I} \otimes \underbrace{\widehat{U}_1}_{k} \otimes \widehat{I} \otimes ... \otimes \widehat{I},$$

где $\widehat{U}^{(k)}$ действует только на k-й кубит посредством гейта $\widehat{U}_1$, преобразующего однокубитовое состояние, квантовый регистр приводится в m-кубитовое состояние, являющееся *когерентной суперпозицией* всех базисных состояний:

$$\widehat{U}^{(m)}\widehat{U}^{(m-1)}...\widehat{U}^{(1)} : |\underbrace{00...0}_{m}\rangle \to \widehat{U}^{(m)}\widehat{U}^{(m-1)}...\widehat{U}^{(1)}|\underbrace{00...0}_{m}\rangle =$$

$$= \sum_{n=0}^{2^m-1} c_n|n\rangle \equiv$$

$$\equiv \sum_{n_{m-1}=0}^{1} \sum_{n_{m-2}=0}^{1} ... \sum_{n_0=0}^{1} c_{n_{m-1}n_{m-2}...n_0}|n_{m-1}n_{m-2}...n_0\rangle, \tag{8}$$

где

$$n = (n_{m-1}n_{m-2}...n_0)_2 = \sum_{l=1}^{m} n_{m-l}2^{m-l}$$

– двоичное представление числа n и

$$\sum_{n=0}^{2^m-1} |c_n|^2 = 1.$$

Состояние (8) является начальным. Ввод информации завершен.

Пример. Если

$$\widehat{U}_1|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad \widehat{U}_1|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle),$$

то

$$\widehat{U}^{(m)}\widehat{U}^{(m-1)}...\widehat{U}^{(1)}|\underbrace{00...0}_{m}\rangle = \frac{1}{\sqrt{2^m}}\sum_{n=0}^{2^m-1}|n\rangle. \tag{9}$$

(Здесь все m-кубитовые базовые состояния $|n\rangle$ равновероятны).

Состояние (9) является начальным. Ввод информации завершен.

5.2. Вычисление

Вычисление – это преобразование $\widehat{U}_F$ начального состояния (8):

$$\widehat{U}_F\left(\sum_{n=0}^{2^m-1}c_n|n\rangle\right) = \sum_{n=0}^{2^m-1}c_n\widehat{U}_F|n\rangle = \sum_{n=0}^{2^m-1}c_n|F(n)\rangle. \tag{10}$$

В случае (9) имеем

$$\widehat{U}_F\left(\frac{1}{\sqrt{2^m}}\sum_{n=0}^{2^m-1}|n\rangle\right) = \frac{1}{\sqrt{2^m}}\sum_{n=0}^{2^m-1}|F(n)\rangle. \tag{11}$$

Конкретная реализация преобразования $\widehat{U}_F$ представляет собой запрограммированный квантовый алгоритм вычисления значений функции F.

Как видно из формулы (10), **в один шаг вычислены сразу все значения функции** F. Это *эффект параллельности* квантовых вычислений, о котором мы говорили в § 2.2.

5.3. Вывод результата

Вывод результата в квантовом компьютеринге – это *измерение* квантового состояния (10):

$$\sum_{n=0}^{2^m-1}c_n|F(n)\rangle \to |F(n)\rangle.$$

В силу принципа квантовой механики вмешательство измеряющего устройства (устройство вывода данных) означает декогеренцию, т.е. разрушение когерентного состояния (10). Мы получаем значение $F(n)$ лишь с вероятностью $|c_n|^2$.

В нашем примере (8) с равной вероятностью $1/2^m$ любое значение $F(n)$!

Получаемый на выходе результат вычислений вследствие декогеренции, как видим, носит *вероятностный характер*! Иначе говоря, то, что получено на выходе, – состояние (регистра) $|F(n)\rangle$ – верно лишь с некоторой вероятностью $|c_n|^2$.

«Наблюдение (части) памяти – не то же самое, что «печать результата». Мы должны спланировать серию прогонов одной и той же квантовой программы и последующую классическую обработку наблюдаемых результатов, и мы можем только надеяться получить желаемый результат с вероятностью, близкой к единице» [3, с.271].

6. Исправление квантовых ошибок

Классические компьютеры надежны, поскольку производимые вычисления можно защитить от *сбоев*, т.е. от ошибок, возникающих вследствие воздействия окружающей среды.

Взаимодействие квантового компьютера с окружающей средой ведет к декогеренции, которая разрушает когерентную суперпозицию и тем самым останавливает то, что делает квантовые вычисления привлекательными по сравнению с классическими, – их параллельность.

Возникновение декогеренции – то же, что появление сбоев в работе классических ЭВМ, поэтому борьбу с декогеренцией, её преодоление называют *исправлением квантовых ошибок.*

Декогеренцию, а также *квантовый шум*, т.е. взаимодействие m-кубита $|q\rangle$ со средой $\mathcal{E}$, можно представить в виде:

$$|q\rangle|\mathcal{E}_0\rangle \to \sum_k \widehat{E}_{i_k}|q\rangle|\mathcal{E}_k\rangle, \tag{12}$$

где

$|\mathcal{E}_0\rangle$ – состояние среды до взаимодействия,
$\widehat{E}_j$ – j-й оператор ошибки (тип ошибки, один из трех, т.е. $j = 1, 2, 3$),
$|\mathcal{E}_k\rangle$ – k-е состояние среды после взаимодействия.

Исправление квантовых ошибок – это процесс Cr, организованный в ходе работы квантового компьютера, который переводит состояния вида $\widehat{E}_{i_k}|q\rangle$ в $|q\rangle$.

В результате имеем восстановление чистого состояния регистра $|q\rangle$:

$$\sum_k \widehat{E}_{i_k}|q\rangle|\mathcal{E}_k\rangle \xrightarrow{Cr} |q\rangle|\mathcal{E}_f\rangle,$$

свободного от помех (сцепленности со средой).

Разработаны различные методы исправления квантовых ошибок.

7. Классический компьютер вычисляет всё, что вычисляет квантовый

Изменения во времени состояния регистра $|\psi(t)\rangle$ квантового компьютера описываются с помощью основного уравнения квантовой механики – уравнения Шрёдингера:

$$\boxed{i\hbar\frac{d}{dt}|\psi(t)\rangle = \widehat{H}|\psi(t)\rangle.}$$

Здесь $\widehat{H}$ – гамильтониан, реализующий конкретный вычислительный (квантовый) алгоритм.

Как известно, решение уравнения Шрёдингера можно записать в виде

$$|\psi(t)\rangle = e^{-\frac{i}{\hbar}\int\limits_0^t \widehat{H}dt}|\psi(0)\rangle.$$

Это квантовая эволюция начального регистра $|\psi(0)\rangle$. Отсюда видно, что найти $|\psi(t)\rangle$ можно, производя классические вычисления экспоненты от матрицы. Это крайне трудоёмкие вычисления, но, в принципе, выполнимые. Следовательно, классический компьютер может вычислить всё, что вычисляет квантовый компьютер, и нет никакого шанса построить квантовый компьютер, вычисляющий классически невычислимые функции.

Литература

1. Валиев К.А., Кокин А.А. Квантовые компьютеры: надежды и реальность. М.: Ижевск: РХД, 2001.
2. Гуц А.К. Основы квантовой кибернетики: Учебное пособие. Омск: Полиграфический центр КАН, 2008. 204 с
3. Манин Ю.И. Классическое и квантовое вычисление и факторизация Шора / Квантовый компьютер и квантовые вычисления. Ижевск: НИЦ «Регулярная и хаотическая динамика», 2001.
4. Манин Ю.И. Вычислимое и невычислимое. М.: Советское радио, 1980.
5. Фейнман Р. Моделирование физики на компьютерах / Сб.: Квантовый компьютер и квантовые вычисления. Ред. ж-ла «Регулярная и хаотическая динамика». Ижевск, 1999. С.96-124.
6. Фейнман Р. Квантовомеханические компьютеры / Сб.: Квантовый компьютер и квантовые вычисления. Ред. ж-ла «Регулярная и хаотическая динамика». Ижевск, 1999. С.125-156.

Математические структуры и моделирование
2010, вып. 21, с. 65–81

УДК 004.93'1

АВТОМАТИЧЕСКАЯ КЛАССИФИКАЦИЯ ТЕКСТОВЫХ ДОКУМЕНТОВ

А.С. Епрев

В данной статье приводится обзор некоторых актуальных методов и подходов, применяемых при решении задач классификации текстовых документов.

Введение

Классификация (категоризация, рубрикация) текстовых документов является задачей автоматического определения документа в одну или несколько категорий (рубрик, тематик) на основании содержания документа. В зарубежной литературе получил широкое распространение термин Text Categorization [1,2].

В настоящее время мы имеем дело с постоянно увеличивающимся объемом обрабатываемой и накапливаемой информации, что делает задачу классификации все более актуальной. Использование классификаторов, позволяет ограничить поиск необходимой информации относительно небольшим подмножеством документов.

Помимо сужения области поиска в поисковых системах задача классификации имеет практическое применение в следующих областях:

- фильтрация спама;
- составление тематических каталогов;
- контекстная реклама;
- системы документооборота;
- автоматический перевод текстов (снятие омонимии).

Данная работа посвящена обзору методов и подходов, применяемых на различных этапах решения задачи классификации текстов.

Омский государственный университет им. Ф.М. Достоевского.
E-mail: anton@eprev.me

1. Формализация задачи

Задача классификации текстов может быть формализована [2] как задача аппроксимации неизвестной функции $\Phi : D \times C \to \{0, 1\}$ (каким образом документы должны быть классифицированы) через функцию $\hat{\Phi} : D \times C \to \{0, 1\}$, именуемую *классификатором*, где $C = \{c_1, \ldots, c_{|C|}\}$ — множество возможных категорий, а $D = \{d_1, \ldots d_{|D|}\}$ — множество документов.

$$\Phi(d_j, c_i) = \begin{cases} 0, & \text{если } d_j \notin c_i; \\ 1, & \text{если } d_j \in c_i. \end{cases}$$

Документ d_j называют *положительным примером* категории c_i, если $\Phi(d_j, c_i) = 1$, и *отрицательным* в противном случае.

Если в задаче каждому документу $d_j \in D$ может соответствовать только одна категория $c_i \in C$, то имеет место *однозначная классификация*, а если произвольное количество категорий $0 < n_j < |C|$ — *многозначная классификация*. Далее будет рассматриваться случай многозначной классификации, если не сказано иного.

Выделяют особый вид классификаторов — *бинарные* (двоичные), множество категорий которых состоит из двух элементов (c_i и его дополнения $\bar{c}_i$). Бинарный классификатор для $\{c_i, \bar{c}_i\}$ определяется функцией $\hat{\Phi}_i : D \to \{0, 1\}$, которая является аппроксимацией неизвестной функции $\Phi_i : D \to \{0, 1\}$.

Нахождение классификатора для множества категорий $C = \{c_1, \ldots, c_{|C|}\}$ обычно рассматривается как поиск $|C|$ бинарных классификаторов $\{c_i, \bar{c}_i\}$, где $i = 1, \ldots, |C|$. Таким образом, классификатор $\hat{\Phi}$ представляет собой множество бинарных классификаторов.

2. Автоматическая классификация

При решении задач автоматической классификации текстовых документов используются методы информационного поиска (Information Retrieval, IR) [3, 4] и машинного обучения (Machine Learning, ML) [4–6].

Документы на есстественном языке преобразовываются в удобную для машинной обработки форму — *индексируются*. В процессе индексирования происходит выделение признаков из документа.

Классификатор для категории c_i автоматически создается в *процессе обучения*, при котором просматривается множество документов с заранее определенными категориями c_i или $\bar{c}_i$ и подбираются такие характеристики классификатора, чтобы новый (ранее не просмотренный) документ, отнесенный к категории c_i, соответствовал им. Чтобы построить классификатор C, необходимо множество документов D, для которых значения функции $\Phi(d_j, c_i)$ известны для каждой пары $\langle d_j, c_i \rangle \in D \times C$.

В экспериментах обычно все множество документов D разделяется на два непересекающихся подмножества [2]:

- набор для обучения (обучающая выборка) $\mathcal{L}$;

- набор для проверки (тестирующая выборка) $\mathcal{T}$.

На обучающем множестве $\mathcal{L}$ строится классификатор и определяются значения его параметров, при которых классификатор выдает лучший результат. На тестовом наборе $\mathcal{T}$ происходит *вычисление эффективности* построенного классификатора. Индексирование документов, построение классификатора и вычисление его эффективности являются темами следующих разделов.

3. Индексирование документов

Индексирование документов в задачах классификации текстов не представлено разнообразием методов и подходов. Обычно документ после индексации представляется как вектор в некотором пространстве (*пространстве признаков*), в котором каждому терму (признаку) ставится в соответствие его вес (значимость):

$$\vec{d_j} = \langle \omega_{1j}, \ldots, \omega_{|T|j} \rangle,$$

где T — словарь, т.е. множество термов, которые встречаются в $|\mathcal{L}|$ обучающих классификатор документах, и $0 \leqslant \omega_{kj} \leqslant 1$ определяет значимость терма t_k в документе d_j.

Подходы к индексированию различаются в методе *определения термов* и способе *вычисления весов*.

Обычно термами являются слова, встречающиеся в документе (за исключением так называемых *стоп-слов*, т.е. нейтральных слов, таких как союзы, предлоги, местоимения и т.п.). Слова, как правило, подвергаются морфологическому разбору или стеммингу (специальный алгоритм для определения морфологического корня слова [7]). В классификации текстов также используется подход к использованию не отдельных слов в качестве термов, а словосочетаний, которые выделяются при помощи синтаксического разбора предложений или статистически, что придает таким термам семантически большую значимость [9, 10].

Вес ω_{kj} может просто определять наличие терма в документе, в таком случае $\omega_{kj} \in \{0, 1\}$. Но обычно в качестве весовых значений используются вещественные числа из диапазона $0 \leqslant \omega_{kj} \leqslant 1$. Такие веса имеют статистическую или вероятностную природу и зависят от метода построения классификатора.

Для определения веса терма можно привлекать дополнительную информацию. Например, если терм находится в заголовке документа, то его вес можно увеличить на несколько процентов. Документы в наборе, как правило, имеют разную длину, поэтому полученные веса принято *нормализовывать*. С обзором подходов к вычислению весов термов можно ознакомиться в статье [11].

3.1. Класс весовых функций $tf * idf$

Наиболее популярным классом статистических весовых функций является $tf * idf$, в котором определены два интуитивных правила: чем чаще терм t_k встречается в документе d_j, тем более значим он в нем (*term frequency*); чем

в большем количестве документов терм t_k встречается, тем менее отличительным он является (*inverse document frequency*). Существует множество вариантов $tf * idf$. Приведем один из них:

$$\omega_{ij} = \frac{tf_{ij} \cdot idf_i}{\sqrt{\sum_k (tf_{kj} \cdot idf_k)^2}},$$

где ω_{ij} — вес i-го терма в документе d_j, tf_{ij} — частота встречаемости i-го терма в рассматриваемом документе (term frequency), $idf_i = \log N/n$ — логарифм отношения количества документов в коллекции к количеству документов, в которых встречается i-ый терм (inverse document frequency). Веса, вычисленные по этой формуле, нормализованы таким образом, что сумма квадратов весов каждого документа равна единице.

4. Уменьшение размерности пространства признаков

Эмпирический закон Хипса [13] связывает рост количества обрабатываемых документов с ростом словаря T. Вычислительная сложность различных методов классификации напрямую зависит от размерности пространства признаков. Поэтому в задачах классификации часто прибегают к сокращению числа используемых термов, т. е. к уменьшению значения $|T|$ до $|T'| \ll |T|$.

Побочным эффектом уменьшения размерности пространства признаков (УР) является *переобучение* (overfitting), когда классификатор слишком хорошо работает на документах из обучающего набора и достаточно плохо на документах, не участвующих в обучении.

Для УР можно прибегнуть к выбору термов из существующих (feature selection) или к созданию искусственных (feature extraction).

4.1. Выбор признаков

Существуют различные методы выбора термов: оставлять наиболее встречающиеся термы в документах или выбирать наиболее значимые, используя различные функции полезности.

Наиболее простой подход к уменьшению размерности пространства признаков заключается в нахождении значений $df(t_k)$ (document frequency — количество документов из $\mathcal{L}$, в которых встречается терм t_k) и выборе наиболее встречающихся. Янг в своей работе [14] показывает возможность уменьшения размерности пространства признаков в 10 раз без потери эффективности и отмечает, что уменьшение в 100 раз приводит к незначительным ухудшениям работы классификатора.

Рассмотрим теперь некоторые функции полезности $f(t_k, c_i)$, характеризующие значимость терма t_k в некотором документе для категории c_i. Чтобы вычислить значимость терма для всей коллекции C, можно, например, найти

среднее значение

$$f(t_k) = \sum_{i=1}^{|C|} P(c_i) f(t_k, c_i)$$

или максимальное

$$f(t_k) = max_{i=1}^{|C|} f(t_k, c_i).$$

Широкое распространение получили функции полезности «прирост информации», «взаимная информация» и «метод хи-квадрат».

Прирост информациии (Information Gain, IG) определяется по формуле

$$IG(t_k, c_i) = \sum_{c \in \{c_i, \bar{c}_i\}} \sum_{t \in \{t_k, \bar{t}_k\}} P(t, c) \cdot \log \frac{P(t, c)}{P(t) \cdot P(c)},$$

где, например, $P(\bar{t}_k, c_i)$ — вероятность того, что терм t_k не встречается в некотором документе d и документ d определен в категорию c_i.

Взаимная информация (Mutual Information, MI):

$$MI(t_k, c_i) = \log \frac{P(t_k, c_i)}{P(t_k) \cdot P(c_i)}.$$

И критерий хи-квадрат:

$$\chi^2 = \frac{|\mathcal{L}| \cdot [P(t_k, c_i) \cdot P(\bar{t}_k, \bar{c}_i) - P(\bar{t}_k, c_i) \cdot P(t_k, \bar{c}_i)]^2}{P(t_k) \cdot P(\bar{t}_k) \cdot P(c_i) \cdot P(\bar{c}_i)}.$$

С подробным сравнением этих методов уменьшения размерности можно ознакомится в статье [14].

4.2. Извлечение признаков

Для уменьшения размерности пространства признаков могут применяться методы кластеризации термов и латентно-семантическое индексирование, в результате которых образуются (извлекаются) новые признаки, способствующие увеличению эффективности классификации [1].

При **кластеризации признаков** происходит объединение в группы термов с высокой попарной семантической близостью, представления этих групп или их центроиды используются в качестве признаков для уменьшения размерности пространства.

Бейкер и Маккалум в своей работе [15] описывают метод кластеризации, при котором уменьшение размерности пространства в 1000 раз приводит к потере эффективности классификации всего на 2%.

В качестве исходных данных в **латентно-семантическом индексировании** (Latent Semantic Indexing, LSI) используется матрица *термы-на-документы*. Столбцы этой матрицы – документы, а строки – термы. Элементами этой матрицы являются веса термов в документах. Задача уменьшения размерности пространства заключается в нахождении сингулярного разложения матрицы.

Разложение матрицы $A \in \mathbb{R}^{m \times n}$ в произведение двух ортогональных матриц $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{n \times n}$ и диагональной матрицы $D = diag(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m \times n}$, где $p = min(m, n)$, называется сингулярным [16]:

$$A = UDV^T.$$

Элементы $\sigma_i \geq 0$ диагональной матрицы D являются корнями собственных чисел матрицы AA^T. Если в матрице D оставить только k наибольших чисел, то произведение полученной диагональной матрицы D' и двух ортогональных U, V является самым близким приближением матрицы A ранга k:

$$A' = UD'V^T.$$

Теперь каждый документ и признак можно представить как линейную комбинацию k линейно-независимых столбцов и строк соответственно.

5. Методы построения классификаторов

В литературе можно встретить различные методы построения классификаторов. Некоторые из них строят двоичные функции $\hat{\Phi} : D \times C \to \{0, 1\}$, а некоторые — вещественные функции $CSV : D \times C \to [0, 1]$ (Categorization Status Value). Если используются первые, то имеет место *точная классификация*, если вторые — *пороговая классификация*. Для последних необходимо определить множество пороговых значений τ_i для $i = 1, \ldots, |C|$ (вычисляются экспериментально на обучающем наборе), которые позволяют рассматривать вещественные значения CSV как двоичные:

$$\hat{\Phi}(d_j, c_i) = \begin{cases} 0, & \text{если } CSV_i(d_j) < \tau_i; \\ 1, & \text{если } CSV_i(d_j) \geqslant \tau_i. \end{cases}$$

Стоит отметить, что в некоторых приложениях вещественные функции могут с успехом использоваться без необходимости преобразования к двоичным. Например, классификаторы с такими функциями могут строить «рейтинг» категорий для документа, просматривая который человек утверждает определенные из них.

Далее будут рассмотрены некоторые из классических методов построения текстовых классификаторов, которые могут служить отправной точкой для разработки более эффективных методик.

5.1. Деревья решений

Классификатор на базе деревьев решений (Decision Trees) [17] для категории c_i представляет собой дерево, узлами которого являются термы t_k, каждое ребро обозначено условием $\geqslant v_k$ или $< v_k$, а листья помечены как c_i или $\bar{c}_i$. Чтобы классифицировать документ d_j в категорию c_i или $\bar{c}_i$, необходимо пройти по

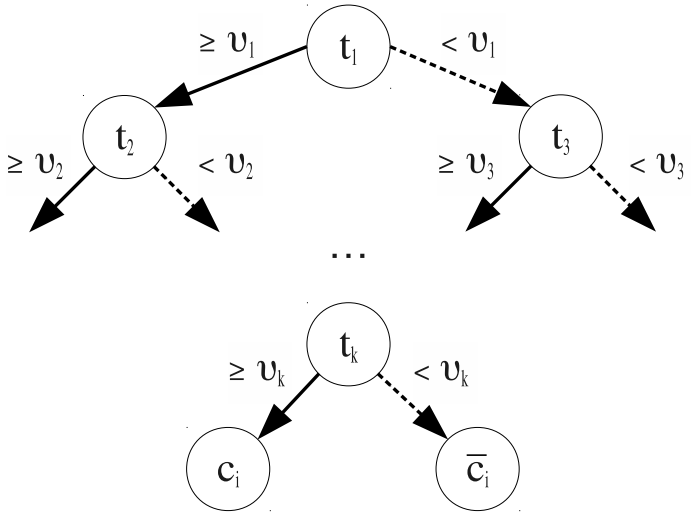


Рис. 1. Дерево решений для категории c_i.

узлам дерева начиная с корня, сравнивая веса терма в документе ω_{kj} со значениями υ_k на ребрах. На практике обычно используют бинарные деревья решений, в которых принятие решения перехода по ребрам осуществляется простой проверкой наличия терма в документе.

Один из способов автоматического построения деревьев решений заключается в последовательном разбиении множества обучающих документов $\mathcal{L}$ на классы, до тех пор пока в классе не останется документов, определенных только в одну из категорий c_i или $\bar{c}_i$. На каждом этапе в качестве узла дерева выбирается терм t_k и определяется υ_k, затем множество документов разбивается на два класса: $\omega_k \geqslant \upsilon_k$ и $\omega_k < \upsilon_k$.

Обычно построенное дерево решений является сильно детализированным (эффект переобучения), поэтому применяются различные алгоритмы усечения дерева. Широкое применение получили алгоритмы ID3 [18] и C4.5 [19].

5.2. Решающие правила

Этот класс классификаторов строит правила классификации вида «если выполняется формула, то категория». В статье [21] можно ознакомиться с системой классификации документов CONSTRUE. Эта система использовалась агентством Reuters для классификации новостных сообщений. Правила классификации составлялись экспертами вручную. Вот одно из таких правил:

```
if   (wheat & farm) or
     (wheat & commodity) or
     (bushels & export) or
     (wheat & tonnes) or
     (wheat & whinter and (¬ soft))
then
     WHEAT
else
     (¬ WHEAT).
```

Одним из способов автоматического построения правил классификации является перебор всевозможных правил в виде формул определенного вида. В статье [22] описывается алгоритм, который применяется при решении задач медицинской направленности. Алгоритм строит формулы вида:

$$C = I_1 \cup I_2 \cup \ldots \cup I_q,$$

где I_i – это конъюнкция p_i признаков. Пространство признаков размерностью порядка нескольких десятков состоит из различных медицинских показателей. Время работы алгоритма экспоненциально зависит от размерности пространства признаков.

Другой подход заключается в построении формул на основе деревьев решений. Каждому пути от корня до листа в дереве решений соответствует правило, где условиями будут являться проверки из узлов, встретившихся на пути. Чтобы получить формулу для категории c_i, нужно объединить все правила для путей, листьями которых является c_i.

5.3. Метод наименьших квадратов

В методе наименьших квадратов (Linear Least-Squares Fit, LLSF) [23] с каждым документом d_j связывают два вектора: входной вектор весов термов $I(d_j)$ размерности $|T|$ и выходной вектор весов категорий $O(d_j)$ размерности $|C|$. Задача классификации сводится к определению вектора $O(d_j)$ по вектору $I(d_j)$:

$$MI(d_j) = O(d_j),$$

где M — матрица размера $|C| \times |T|$.

Построение классификатора заключается в нахождении матрицы $\hat{M}$ минимизирующей норму $\|\hat{M}I - O\|_F$, где I — матрица размера $|T| \times |\mathcal{L}|$, столбцы которой являются векторами $I(d_j)$, O — матрица размера $|C| \times |\mathcal{L}|$, столбцы которой являются векторами $O(d_j)$, и $\|V\|_F$ — норма Фробениуса для матрицы размера $n \times m$:

$$\|V\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} v_{ij}^2}.$$

Метод наименьших квадратов позволяет минимизировать корень из суммы квадратов всех ошибок при обучении классификатора. Матрицу $\hat{M}$ получают

через сингулярное разложение матрицы, построенной на обучающем множестве, а элементы m_{ik} определяют связь между категорией c_i и термом t_k.

5.4. Адаптивные линейные классификаторы

В классе линейных классификаторов категории и документы представлены векторами $\vec{c}_i = \langle \omega_{1i}, \ldots, \omega_{|T|i} \rangle$ и $\vec{d}_j = \langle \omega_{1j}, \ldots, \omega_{|T|j} \rangle$ соответственно. В качестве $CSV_i(d_j)$ используется значение косинуса угла между векторами $\vec{d}_j$ и $\vec{c}_i$:

$$cos(\vec{d}_j, \vec{c}_i) = \frac{\sum\limits_{k=1}^{|T|} \omega_{ki} \cdot \omega_{kj}}{\sqrt{\sum\limits_{k=1}^{|T|} \omega_{ki}^2} \cdot \sqrt{\sum\limits_{k=1}^{|T|} \omega_{kj}^2}}.$$

Адаптивные линейные классификатора (On-Line Linear Classifiers) [24–26] осуществляют построение классификатора на первом просмотренном документе и постоянно совершенствуют его на последующих.

Один из методов автоматического построения классификатора для категории c_i заключается в следующем. Изначально $\vec{c}_i = \langle 1, \ldots, 1 \rangle$. Затем для каждого документа $d_j \in \mathcal{L}$, представленного вектором $\vec{d}_j$ с двоичными весами, вычисляется $CSV_i(d_j)$ и вносятся поправки в $\vec{c}_i$, если результат классификации неверен : если $d_j \in c_i$, тогда веса ω_{ki} для k, таких, что $\omega_{kj} = 1$, увеличивают на некоторую фиксированную величину $\alpha > 0$ и понижают на эту же величину в том случае, если $d_j \notin c_i$.

Такой подход позволяет продолжить обучение классификатора после его построения. Кроме того, термы t_k с маленькими весами ω_{ki}, которые не оказывают влияния на результаты классификации, можно выбросить из рассмотрения, тем самым уменьшить размерность пространства признаков.

5.5. Метод Rocchio

Использовать метод Rocchio [27] для построения линейного классификатора впервые было предложено в работе [28]. Для каждой категории c_i вычисляем вектор $\vec{c}_i = \langle \omega_{1i}, \ldots, \omega_{|T|i} \rangle$ по формуле

$$\omega_{ki} = \beta \cdot \sum_{d_j \in D_i^+} \frac{\omega_{kj}}{|D_i^+|} - \gamma \cdot \sum_{d_j \in D_i^-} \frac{\omega_{kj}}{|D_i^-|},$$

где ω_{kj} — вес терма t_k в документе d_j, $D_i^+ = \{d \in \mathcal{L} \mid \Phi(d_j, c_i) = 1\}$ и $D_i^- = \{d \in \mathcal{L} \mid \Phi(d_j, c_i) = 0\}$. Параметры β и γ определяют значимость положительных и отрицательных примеров. В случае, когда $\beta = 1$ и $\gamma = 0$, вектор $\vec{c}_i$ будет являться центроидом положительных примеров категории c_i.

5.6. Метод k-NN

Метод k-ближайших соседей (k-Nearest Neighbours, k-NN) [29] в отличие от других не требует обучения. Для того чтобы найти категории, соответствующие документу d_j, классификатор сравнивает d_j со всеми документами из обучающей выборки $\mathcal{L}$: для каждого $d_z \in \mathcal{L}$ вычисляется «расстояние» $\rho(d_j, d_z)$. Далее из обучающей выборки выбираются k документов, ближайших к d_j. Для категорий вычисляются функции ранжирования $CSV_i(d_j)$ по формуле

$$\sum_{d_z \in \mathcal{L}_k(d_j)} \rho(d_j, d_z) \cdot \Phi(d_z, c_i),$$

где $\mathcal{L}_k(d_j)$ — это ближайшие k документов из $\mathcal{L}$ к d_j.

Параметр k обычно выбирается в интервале $20 \ldots 50$. Документ d_j определен в категории, для которых $CSV_i(d_j) \geqslant \tau_i$. Данный метод дает высокую эффективность, но при этом требователен к вычислительным ресурсам на этапе классификации.

5.7. Метод опорных векторов

Метод опорных векторов (Support Vector Machine, SVM) [31] заключается в нахождении гиперплоскости в пространстве признаков $\mathbb{R}^{|T|}$, разделяющей его на две части: положительные примеры в одной и отрицательные в другой — у которой минимальное расстояние до ближайших примеров максимально.

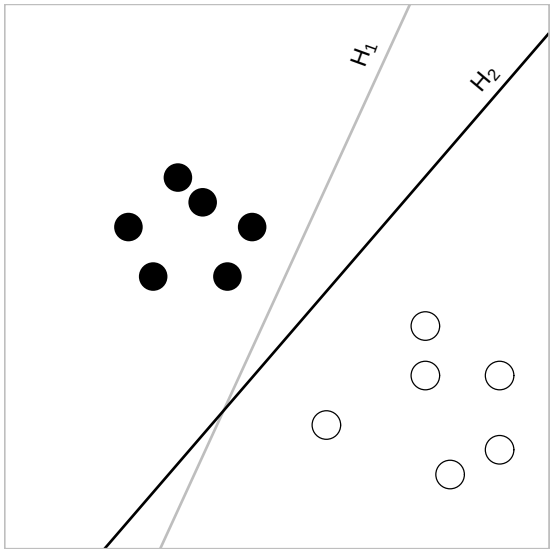


Рис. 2. Метод опорных векторов. Две классифицирующих разделяющих прямых (гиперплоскости), но только H_2 разделяет два множества с большим отступом

В том случае, когда такой разделяющей гиперплоскости не существуют (проблема линейной неразделимости), используют подход, заключающийся в пе-

реходе от исходного пространства признаков к новому, в котором обучающая выборка окажется линейно разделимой.

Для классификаторов на базе метода опорных векторов, как правило, не требуется уменьшать размерность пространства признаков; они довольно устойчивы к переобучению и хорошо масштабируются [31]. Подробную информацию о методе опорных векторов можно найти в работах [32, 33].

5.8. Метод Байеса

В вероятностном классификаторе [34] используется векторное представление документов, а функции $CSV_i(d_j)$ рассматриваются в терминах условных вероятностей $P(c_i|\vec{d_j})$ (вероятность того, что документ, представленный вектором $\vec{d_j}$, соответствует категории c_i). Работа вероятностного классификатора заключается в вычислении значений $P(c_i|\vec{d_j})$ для $i = 1 \dots |C|$ и нахождении наибольшей такой вероятности:

$$H(\vec{d_j}) = \arg\max_{c_i \in C} P(c_i|\vec{d_j}).$$

Условную вероятность $P(c_i|\vec{d_j})$ согласно теореме Байеса можно переписать как

$$P(c_i|\vec{d_j}) = \frac{P(\vec{d_j}|c_i) \cdot P(c_i)}{P(\vec{d_j})}, \tag{1}$$

где $P(c_i)$ — это априорная вероятность того, что документ определен в категорию c_i, $P(\vec{d_j}|c_i)$ — вероятность найти документ, представленный вектором $\vec{d_j}$, в категории c_i и $P(\vec{d_j})$ — вероятность того, что случайно выбранный документ будет иметь вектор $\vec{d_j}$.

По сути $P(c_i)$ является отношением количества документов из обучающей выборки $\mathcal{L}$, отнесенных в категорию c_i, к количеству всех документов из $\mathcal{L}$:

$$P(c_i) = \frac{|\{d_j \in \mathcal{L} \mid d_j \in c_i\}|}{|\mathcal{L}|}.$$

Чтобы вычислить $P(\vec{d_j}|c_i)$ (и $P(\vec{d_j})$) необходимо сделать допущение о том, что вхождение термов в документ зависит от категории, но не зависит от других термов этого документа. Таким образом, $P(\vec{d_j}|c_i)$ можно записать как

$$P(\vec{d_j}|c_i) = \prod_{k=1}^{|T|} P(\omega_{kj}|c_i).$$

В свою очередь $P(\omega_{kj}|c_i)$ можно определить как отношение количества документов из обучающей выборки $\mathcal{L}$, отнесенных в категорию c_i и содержащих терм t_k, к общему количеству всех документов из $\mathcal{L}$, отнесенных в категорию c_i:

$$P(\omega_{kj}|c_i) = \frac{|\{d_j \in \mathcal{L} \mid d_j \in c_i, t_k \in d_j\}|}{|\{d_j \in \mathcal{L} \mid d_j \in c_i\}|}.$$

Из-за предположения о независимости признаков такой классификатор называют «наивный байесовский» (Naive Bayes Classifier) [9, 20, 31, 35].

5.9. Другие методы

В литературе можно встретить и другие методы построения классификаторов, такие как байесовские [36,37] и нейронные [24–26,38–40] сети, методы с использованием генетических алгоритмов [41,42] и N-грамм [43].

6. Оценка эффективности

Эффективность классификатора $\hat{\Phi}_i$ является качественной оценкой результатов его работы на тестирующей выборке $\mathcal{T}$. Эффективность используется для сравнения различных методов классификации.

При однозначной классификации эффективность классификатора можно рассматривать как отношение верно классифицированных документов к общему количеству документов. Но в случае использования бинарных классификаторов (при многозначной классификации) такое отношение не пригодно для оценки эффективности. Причиной этому является то, что категории c_i и $\bar{c}_i$ обычно не сбалансированы, то есть одна из них содержит намного больше документов, чем другая. В таком случае построение классификатора, который бы давал высокую эффективность, является тривиальной задачей — достаточно, чтобы классификатор сопоставлял всем документам наиболее часто встречающуюся категорию. Но о практической применимости такого классификатора не может быть и речи.

Как результат, для бинарных классификаторов часто применяется оценка эффективности как комбинация *точности* (precision) и *полноты* (recall). Точность p_i — доля верно классифицированных в c_i документов, а полнота r_i — отношение верно классифицированных в c_i документов к общему количеству документов, которые должны были быть классифицированы в c_i.

Например, в категорию c_i было отнесено 10 документов, из них только восемь оказались правильно классифицированы. Таким образом, точность $p_i = 8/10$. Но в тестирующей выборке было 12 документов, отнесенных в c_i, получаем, что $r_i = 8/12$.

В задачах многозначной классификации эффективность, которая заключается в вычислении точности и полноты для каждой категории индивидуально, необходимо усреднить. Существует два подхода усреднения (таблица 1): микроусреднение (которое учитывает «вес» категории) и макроусреднение (для которого все категории равны). Наиболее часто используется макроподход, потому что именно он позволяет рассмотреть категории независимо от их встречаемости в корпусе документов.

Классификаторы можно настроить на увеличение точности в ущерб простоте (и наоборот) изменением значений τ_i в $CSV_i : D \to \{0, 1\}$ [1]. Поэтому только комбинация точности и полноты может служить хорошей оценкой эффективности. Наиболее популярным способом объединения этих двух оценок является функция

$$F_\beta = \frac{(\beta^2 + 1)pr}{\beta^2 p + r},$$

Таблица 1. Усреднение точности и полноты; $D_i = \{d_j \in \mathcal{T} \mid \Phi(d_j, c_i) = 1\}$, $\hat{D}_i = \{d_j \in \mathcal{T} \mid \hat{\Phi}(d_j, c_i) = 1\}$

	Точность, p	Полнота, r
Микроусреднение	$p = \dfrac{\sum_{i=1}^{\lvert C\rvert} \lvert \hat{D}_i \cap D_i\rvert}{\sum_{i=1}^{\lvert C\rvert} \lvert \hat{D}_i\rvert}$	$r = \dfrac{\sum_{i=1}^{\lvert C\rvert} \lvert \hat{D}_i \cap D_i\rvert}{\sum_{i=1}^{\lvert C\rvert} \lvert D_i\rvert}$
Макроусреднение	$p = \dfrac{1}{\lvert C\rvert} \sum_{i=1}^{\lvert C\rvert} \dfrac{\lvert \hat{D}_i \cap D_i\rvert}{\lvert \hat{D}_i\rvert}$	$r = \dfrac{1}{\lvert C\rvert} \sum_{i=1}^{\lvert C\rvert} \dfrac{\lvert \hat{D}_i \cap D_i\rvert}{\lvert D_i\rvert}$

где $0 \leqslant \beta \leqslant \infty$. Обычно в качестве значения β используется единица, и функция F_β принимает, вид в котором точность и полнота находятся в равных весовых категориях:

$$F_1 = \frac{2pr}{p + r}.$$

Заметим, что для классификатора, который назначает всем документам часто встречающуюся категорию, $p = 1$ и $r = 0$ таким образом, $F_\beta = 0$ для любого β. Аналогично и для классификатора, который всем документам будет назначать все категории, в таком случае $p = 0$ и $r = 1$, а $F_\beta = 0$ для любых β.

В некоторых приложениях классификации текстов, таких как фильтрация спама (бинарный классификатор для определения электронного письма в одну из двух категорий – «спам» и его дополнения «не спам»), точность имеет большее значение, чем полнота, потому что отнесение «хорошего» письма в категорию «спам» является большей ошибкой, чем принятие нежелательного сообщения как «не спам». Оценкой эффективности может служить F_β, где $0 \leqslant \beta < 1$, что позволяет больше внимания уделять точности чем полноте. А выбором значения $1 < \beta < \infty$ внимание уделяетеся полноте в ущерб точности.

7. Сравнение классификаторов

Сравнение методов построения классификаторов является довольно сложной задачей по причине того, что разные входные данные могут приводить к различным результатам. Чтобы провести сравнение различных классификаторов, необходимо выполнить их построение и вычисление эффективности на одинаковых наборах документов для обучения и тестирования. Широкое распространение получил корпус текстов Reuters-21578 [44], для которого существуют фиксированные разбиения на обучающее и тестирующее множества.

Таблица 2. Сравнение эффективности различных классификаторов [40]

Метод	Микро r	Микро p	Микро F_1	Макро F_1
Метод опорных векторов	.8120	.9137	.8599	.5251
Метод k-ближайших соседей	.8339	.8807	.8567	.5442
Метод наименьших квадратов	.8507	.8489	.8498	.5008
Нейронная сеть	.7842	.8785	.8287	.3765
Метод Байеса	.7688	.8245	.7956	.3886

В таблице 2 приведены результаты эксперименто, опубликованных в работе [40]. Построение классификаторов и оценка их эффективности проводилась с использованием разбиения «ModApte» коллекции документов Reuters-21578. Это разбиение задает 90 категорий, 9603 документа содержатся в обучающем наборе и 3299 документов в тестирующем.

С результатами других экспериментов можно ознакомиться в [10] (сравниваются байесовские сети, деревья решений, методы Байеса и опорных векторов) и [31] (сравниваются методы Баейса, Rocchio, k-ближайших соседей, опорных векторов и деревья решений). Автор статьи [40] отмечает, что его результаты немного отличаются от опубликованных в [31], но классификатор, построенный на базе метода опорных векторов, также имеет небольшое преимущество перед остальными.

8. Ансамбли классификаторов

Использование комбинации классификаторов позволяет повысить точность классификации [45]. Идея заключается в построении k классификаторов $\hat{\Phi}_1, \ldots, \hat{\Phi}_k$ и объединении их результатов классификации. В машинном обучении широкое распространение получили методы «bagging» и «boosting» [46], которые основаны на изменении обучающего множества.

В методе «bagging» построение k классификаторов $\hat{\Phi}_i$ осуществляется независимо друг от друга на обучающих множествах, полученных из исходного случайной заменой документов (размер обучающего множества остается прежним, просто одни документы отсутствуют, а другие встречаются несколько раз). Результат классификации определяется простым большинством голосов элементов ансамбля.

Идея метода «boosting» заключается в последовательном построении k классификаторов, при котором на классификатор $\hat{\Phi}_i$ оказывают влияние $\hat{\Phi}_1, \ldots, \hat{\Phi}_{i-1}$. Классификатор $\hat{\Phi}_i$ строится на исходном обучающем множестве, документы d_j которого участвуют в обучении с некоторыми весовыми коэффициентами h_j^i. После обучения классификатор $\hat{\Phi}_i$ проверяется на исходной обучающей выборке и происходит пересчет коэффициентов. Коэффициент h_j^{i+1} уменьшается, если документ d_j классифицирован верно, и увеличивается в противном случае. В методе «boosting» используется взвешенная линейная комби-

нация голосов элементов ансамбля.

В работе [47] приводятся теоретическое обоснование и результаты эксперимента, которые показывают, что комбинации независимых классификаторов наиболее эффективны. В последнее время большую популярность получили методы, в которых обучение отдельных элементов ансамбля осуществляется независимо на различающихся подмножествах признаков [48, 49].

Литература

1. Sebastiani F. Machine Learning in Automated Text Categorization // ACM Computing Surveys. 2002. V. 34, N. 1. P. 1–47.
2. Sebastiani F. Text Categorization // Text Mining and Its Applications. WIT Press, Southampton, UK, 2005. P. 109–129.
3. Manning C., Raghavan P., Schutze H. Introduction to Information Retrieval. Cambridge University Press, 2008. 544 p.
4. Adam Berger. Statistical Machine Learning for Information Retrieval. Carnegie Mellon University, 2001. 143 p.
5. Witten I. H., Frank E. Data Mining: Practical Machine Learning Tools and Techniques (Second Edition). Morgan Kaufmann, 2005. 525 p.
6. Paliouras G., Karkaletsis V., Spyropoulos C. D. Machine Learning and Its Applications: Advanced Lectures (Lecture Notes in Computer Science / Lecture Notes in Artificial Intelligence). Springer, 2001. 325 p.
7. Bill Frakes. Stemming algorithms // Information Retrieval: Data Structures and Algorithms. Englewood Cliffs, US. 1992. P. 131–160.
8. Thorsten Joachims. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization // Proceedings of International Conference on Machine Learning (ICML). 1997. 26 p.
9. Lewis D.D. An evaluation of phrasal and clustered representations on a text categorization task // Proceedings of SIGIR-92, 15th ACM International Conference on Research and Development in Information Retrieval. ACM Press, US. 1992. P. 37–50.
10. Dumais S.T., Platt J., Heckerman D., Sahami M. Inductive learning algorithms and representations for text categorization // Proceedings of CIKM-98, 7th ACM International Conference on Information and Knowledge Management, Bethesda, MD. 1998. P. 148–155.
11. Salton G, Buckley C. Term-Weighting Approaches in Automatic Text Retrieval // Information Processing and Management. 1988. P. 513–523.
12. Lewis D.D., Ringuette M. A comparison of two learning algorithms for text categorization // Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, Las Vegas, US. 1994. P. 81–93.
13. Heaps H.S. Information Retrieval: Computational and Theoretical Aspects. Academic Press, 1978. 368 p.
14. Yang Y. Pedersen J.O. A comparative study on feature selection in text categorization // Proceedings of ICML-97, 14th International Conference on Machine Learning. Morgan Kaufmann Publishers, San Francisco, US: Nashville, US. 1997. P. 412–420.

15. Baker L.D., McCallum A.K. Distributional clustering of words for text classification // Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval, Melbourne, Australia. 1998. P. 96–103.
16. Meltzer T. SVD and its Application to Generalized Eigenvalue Problems. 2004. 16 p.
17. Mitchell T. M. Machine Learning. McGraw Hill, New York, 1997. 414 p.
18. Quinlan J. Induction of decision trees // Machine Learning. 1998. V. 1, N. 1. P. 81–106.
19. Quinlan J. C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993. 302 p.
20. Li Y. H., Jain A. K. Classification of Text Documents // The Computer Journal. 1998. V. 41, N. 8. P. 537–546.
21. Hayes P.J., Weinstein S.P. Construe: A System for Content-Based Indexing of a Database of News Stories // Proceedings of the Second Annual Conference on Innovative Applications of Intelligence. 1990.
22. Marshall R.J. Generation of Boolean classification rules // Proceedings of Computational Statistics, Utrecht, The Netherlands. 2000. P. 355–360.
23. Yang Y., Chute C. G. An example-based mapping method for text categorization and retrieval // ACM Trans. Inform. Syst. 1994. V. 12, N. 3. P. 252–277.
24. Schutze H., Hull D. A., Pedersen J. O. A comparison of classifiers and document representations for the routing problem // Proceedings of SIGIR-95, 18th ACM International Conference on Research and Development in Information Retrieval, Seattle. 1995. P. 229–237.
25. Ng H. T., Goh W. B., Low K. L. Feature selection, perceptron learning, and a usability case study for text categorization // Proceedings of SIGIR-97, 20th ACM International Conference on Research and Development in Information Retrieval, Philadelphia. 1997. P. 67–73.
26. Dagan I., Karov Y., Roth D. Mistake-driven learning in text categorization // Proceedings of EMNLP-97, 2nd Conference on Empirical Methods in Natural Language Processing, Providence, RI. 1997. P. 55–63.
27. Rocchio J.J. Relevance feedback in information retrieval // The SMART Retrieval System: Experiments in Automatic Document Processing. 1971. P. 313–323.
28. Hull D. A. Improving text retrieval for the routing problem using latent semantic indexing // Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval, Dublin, Ireland. 1994. P. 282–289.
29. Yang Y. Expert network: effective and efficient learning from human decisions in text categorisation and retrieval // Proceedings of SGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval, Dublin, Ireland. 1994. P. 13–22.
30. Vapnik V., The Nature of Statistical Learning Theory. Springer-Verlag, 1995. 188 p.
31. Joachims T. Text categorization with support vector machines: learning with many relevant features // Proceedings of ECML-98, 10th European Conference on Machine Learning, Chemnitz, Germany. 1998. P. 137–142.
32. Воронцов К. В. Лекции по методу опорных векторов. 2007. 18 с. URL: http://www.ccas.ru/voron/download/SVM.pdf (дата обращения: 12.12.2009)
33. Cristianini N., Shawe-Taulor J. An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press, 2000. 189 p.
34. Lewis, D.D. Naive (Bayes) at forty: The independence assumption in information retrieval // Proceedings of ECML-98, 10th European Conference on Machine Learning,

Chemnitz, Germany. 1998. P. 4–15.
35. Koller D., Sahami M. Hierarchically classifying documents using very few words // Proceedings of ICML-97, 14th International Conference on Machine Learning, Nashville. 1997. P. 170–178.
36. Heckerman D. A Tutorial on Learning With Bayesian Networks // Learning in graphical models. 1999. P. 301–354.
37. L. M. de Campos, A. E. Romero. Bayesian network models for hierarchical text classification from a thesaurus // International Journal of Approximate Reasoning. 2009. V. 50, N. 7. P. 932–944.
38. Lam S.L., Lee D.L. Feature reduction for neural network based text categorization // Proceedings of DASFAA-99, Taiwan. 1999. P. 195–202.
39. Ruiz M., Srinivasan P. Hierarchical Text Categorization Using Neural Networks // Information Retrieval. 2002. V. 5, N. 1. P. 87–118.
40. Yang Y., Liu X. A re-examination of text categorization methods // Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval, Berkeley, CA. 1999. P. 42–49.
41. Wong M. L., Cheung K. S. Data Mining Using Grammar Based Genetic Programming and Applications. Kluwer Academic Publishers, 2002. 228 p.
42. Lankhorst M. Automatic Word Categorization with Genetic Algorithms // Proceedings of the ECAI'94 Workshop on Applied Genetic and other Evolutionary Algorithms. 1994.
43. Cavnar W. B., Trenkle J. M. N-Gram-Based Text Categorization // Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval. 1994. P. 161–175.
44. Lewis D. The Reuters-21578 text categorizationtest collection. 1999. URL: http://www.daviddlewis.com/resources/testcollections/reuters21578/ (дата обращения: 12.12.2009)
45. Dietterich T. G. Machine learning research: four current directions // AI Magazine. 1997. V. 18. P. 97–136.
46. Quinlan J. R. Bagging, Boosting, and C4.5 // Proceedings of AAA/IAAI. 1996. P. 725–730.
47. Oza N. C., Tumer K. Decimated input ensembles for improved generalization // Proceedings of the International Joint Conference on Neural Networks, Washington, DC. 1999.
48. Bryll R. Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets // Pattern Recognition. 2003. V. 36. P. 1291–1302.
49. Bay S. D. Nearest neighbor classifiers from multiple feature subsets // Intelligent data analysis. 1999. V. 3. P. 191–209.

Математические структуры и моделирование
2010, вып. 21, с. 82–95

УДК 004.92

АЛГОРИТМЫ ОБНАРУЖЕНИЯ СТОЛКНОВЕНИЙ

Д.И. Собинов, В.В. Коробицын

В статье приводится обзор современных алгоритмов и подходов для построения систем обнаружения столкновений объектов. Приводится классификация алгоритмов по нескольким признакам.

1. Введение

В настоящее время, в связи со значительным прогрессом в области разработки аппаратного обеспечения вычислительной техники, стало возможным осуществлять моделирование движения реального мира в соответствии с физическими законами. Результаты такого рода моделирования используются в различных областях человеческой деятельности: при разработке систем виртуальной реальности и тренажеров, компьютерных игр, систем автоматизированного проектирования, в робототехнике и т.д. Базовой частью подобных систем является подсистема обнаружения столкновений. В общем случае, задача данной подсистемы состоит в проверке простого факта: пересекаются ли два объекта в пространстве. В случае, если объекты пересекаются, часто бывает необходима дополнительная информация, такая как нахождение объема пересечения, его аппроксимация в виде набора точек или простого геометрического объекта, глубина взаимопроникновения объектов. Эта информация используется, например, для оценки силы, которая должна быть приложена к моделируемым телам для их расталкивания в соответствии с законами механики.

Усилия исследователей в рассматриваемой области сосредоточены на решении следующих задач:

- создание универсальных алгоритмов и подходов для прозрачной работы с произвольными объектами: выпуклыми и вогнутыми, деформируемыми и абсолютно твердыми телами, статическими и движущимися и т.д.;
- повышение эффективности алгоритмов, получение максимальной производительности с учетом аппаратной платформы, на которой будет эксплуатироваться программная система;
- повышение устойчивости результатов;

Омский государственный университет им. Ф.М. Достоевского.
E-mail: dsobinov@gmail.com

- упрощение алгоритмов при сохранении других качественных параметров, что позволяет уменьшить стоимость реализации подсистем обнаружения столкновений.

За последние годы исследователями было предложено большое количество разнообразных алгоритмов. Были даже изданы книги [1, 2], обобщающие накопленный опыт в построении систем обнаружения столкновений. Целью данной статьи является общая классификация подходов и алгоритмов определения столкновений и обзор тех, что появились за последние несколько лет.

2. Классификация алгоритмов определения столкновений

2.1. По представлению входных данных

Алгоритмы, использующиеся в системах обнаружения столкновений, могут быть классифицированы по способу представления объектов на входе системы. Наиболее часто встречаются входные данные следующих типов:

- сетки из треугольников;
- конструктивная блочная геометрия;
- неявно заданная геометрия.

Использование первого типа обусловлено тем, что треугольник является базовым геометрическим примитивом для отображения на экране дисплея сложных объектов. Координаты точек передаются через API драйверу графического адаптера, который растеризует входные данные и выводит на экран.

Конструктивная блочная геометрия часто используется в САПР. Комбинируя простые базовые трехмерные объекты, такие как сферы, прямоугольные параллелепипеды, конусы с помощью булевых операций, на выходе получаем сложный объект.

Неявное задание объекта предполагает задание формы объекта в виде математического уравнения. Например, область, ограниченную сферой, можно задать в виде уравнения $x^2 + y^2 + z^2 \leq r^2$. Очевидно, что этот способ слишком громоздкий и не годится для сложных объектов.

Далее, если не указано явно, предполагается, что в качестве входных данных для алгоритма обнаружения столкновений используются полигональные сетки из треугольников.

2.2. По связям с системой моделирования

Чаще всего система обнаружения столкновений является подсистемой какой-либо более крупной системы, например моделирования хирургической операции. При этом подсистема определения столкновений может либо являться «черным ящиком» для остальных компонентов, либо быть тесно интегрированной с остальными компонентами.

В первом случае такая подсистема может разрабатываться независимо от других компонентов системы в качестве сторонней библиотеки. Преимуществом подобного подхода является возможность повторного использования наработанного программного кода. Недостатком же можно считать сложность разработки и невозможность учесть все требования в каждом конкретном случае использования.

При разработке подсистемы обнаружения столкновений может учитываться специфика проекта, например необходимость в использовании особого формата задания объектов, или тесная интеграция системы с компонентом моделирования, например обнаружение столкновений и одновременное вычисление расталкивающей силы для двух тел [3].

Для этого случая могут применяться специализированные алгоритмы, которые позволяют добиться лучшей производительности и устойчивости.

2.3. Классификация по фазам

Наиболее простым методом для обнаружения столкновений между двумя объектами A и B является попарная проверка всех элементов, составляющих A, со всеми элементами из B. Учитывая, что на каждом шаге моделирования может участвовать большое количество объектов, состоящих из тысяч многоугольников, можно заключить, что обнаружение столкновений является вычислительно сложной задачей. Если сразу проводить детальную проверку на пересечение с нахождением всей необходимой информации (характеристики объема пересечения, глубина взаимопроникновения и т.д.), то в итоге система будет обладать низкой масштабируемостью и уже при сравнительно небольшом количестве моделируемых объектов скорость работы не будет удовлетворять жестким условиям систем реального времени.

В связи с этим алгоритмы определения столкновений часто делят на последовательность шагов, или фаз. Одно из первых упоминаний такого разделения приводится в работе [4]. Широкая фаза отсеивает те пары объектов (или элементов одного объекта), которые заведомо не пересекаются. Узкая фаза необходима для детального рассмотрения каждой из пар объектов, не отброшенных при выполнении широкой фазы. На данном этапе, если объекты пересекаются, то для них находится контактная информация.

Соответственно, наиболее общим признаком, по которому можно классифицировать алгоритмы определения столкновений, будет принадлежность к широкой или узкой фазе.

2.3.1. Широкая фаза

Для широкой фазы подходят алгоритмы, которые позволяют с минимальными временными затратами отбраковать наибольшее количество пар. При этом преимущество отдается скорости работы алгоритма. Для этого вместо сложного исходного геометрического представления объекта используют ограничивающую геометрию, или ограничивающий объем. Ограничивающий объем — это область пространства, которая содержит данный объект. Для широкой фазы

используют простые по форме объекты, чаще всего сферы или прямоугольные параллелепипеды. Габариты ограничивающего объема должны быть такими, чтобы исходный объект целиком помещался в него, но при этом имел минимально возможный объем.

Таким образом, простая попарная проверка друг с другом объектов с большим количеством элементов заменяется на проверку на пересечение между простыми объектами. Например, для определения, пересекаются ли две сферы, необходимо оценить истинность следующего выражения:

$$R_1 + R_2 \geq |O_1 - O_2|,$$

где R_1 и R_2 – радиусы сфер, O_1 и O_2 – радиус-векторы их центров.

Если два ограничивающих объема пересекаются, то далее производится их проверка на пересечение с использованием исходного геометрического представления (т.е. происходит передача пары объектов на узкую фазу). В реальных моделируемых сценах лишь небольшое количество объектов находится в непосредственной близости, следовательно, получается заметный выигрыш в производительности.

Дальнейшее увеличение производительности можно получить, если вместо наиболее простого случая попарной проверки всех моделируемых объектов применить либо техники пространственного разделения сцены, либо алгоритмы, учитывающие когерентность временных шагов моделирования, т.е. когда положения тел изменяются не слишком сильно по прошествии малого временного шага. К последней группе относится алгоритм «sweep-and-prune», впервые описанный в [5]. Основная идея алгоритма — проецирование объектов сцены на одну или несколько координатных осей, сохранение минимальных и максимальных проекций для объектов в списки и далее на каждом временном шаге поддержании списков в упорядоченном состоянии. Если временная когерентность высокая, то затраты на сортировку будут незначительными при использовании подходящих методов, например сортировки вставками. Запрос на столкновение для объекта может быть выполнен за время, близкое к константе путем проверки соседних значений из списка и получение соответствующих им объектов. В работе [6] приводится целый ряд улучшений данного алгоритма, позволяющих повысить его эффективность. Авторами была проведена оценка, насколько то или иное улучшение влияет на производительность для различных конфигураций моделируемой сцены.

Пространственное разделение предполагает проверку на пересечение только тех объектов, которые попали в один пространственный регион. Один из способов построения разбиения предполагает рекурсивное деление сцены, пока соблюдается некоторое условие, например размер региона, содержащегося в листовом узле, не слишком мал.

Наиболее простой вариант разбиения — равномерное деление плоскости на двумерные ячейки одного размера и далее проецирование всех объектов сцены на эту плоскость. Таким образом, каждый объект ассоциирован с одной или несколькими ячейками. Чтобы найти объекты, потенциально пересекающиеся с заданным, необходимо проверить только те объекты, которые находятся в той

же ячейке, что и заданный. В [7] показано, как подобный алгоритм может быть реализован на CUDA на GPU со значительным выигрышем в производительности по сравнению с версией на CPU. Основным недостатком равномерного разбиения пространства является неэффективность проверок для случая, когда сцена содержит объекты с сильно различающимися размерами. В этом случае сложно подобрать подходящий размер ячейки.

Также в качестве схемы разбиения пространства можно использовать различные иерархические структуры: восьмеричные деревья или квадродеревья, иерархическое разбиение на ячейки [1]. В работе [8] были использованы деревья бинарного разбиения (Binary Space Partitioning) и предложен алгоритм эффективного обновления структуры такого дерева при изменении позиций объектов сцены с течением времени. Авторам удалось найти компромисс между затратами на нахождение потенциально пересекающихся пар объектов и количеством таких пар на выходе широкой фазы.

2.3.2. Узкая фаза

Алгоритм узкой фазы принимает на вход список пар объектов, отобранных широкой фазой как наиболее вероятных кандидатов на столкновение. Используя исходное геометрическое представление объектов, алгоритм на выходе дает ответ на вопрос, пересекаются ли на самом деле объекты, поданные на вход. Если пара объектов пересекается, алгоритм также может вернуть дополнительную информацию:

- глубину взаимопроникновения объектов друг в друга, т.е. минимальное расстояние, на которое нужно переместить объекты, чтобы они перестали пересекаться;
- нормаль, вдоль которой считается глубина взаимопроникновения;
- аппроксимацию объема пересечения.

Эта информация может использоваться для нахождения времени, когда объекты имели одну точку соприкосновения, для вычисления импульса, который необходимо приложить к объектам для их реалистичного расталкивания и т.д.

Алгоритмы узкой фазы можно разделить на две группы по характеру получения результата:

- работающие в пространстве объектов;
- работающие в пространстве изображений.

Работающие в пространстве изображений алгоритмы характеризуются тем, что используют графические операции, такие как растеризация, для обнаружения столкновений. Таким образом, объекты сцены отображаются в пространство с ограниченной размерностью, например в буфер кадра или LDI (Layered Depth Image, [9]), и, следовательно, столкновение находится с погрешностью, зависящей от размерности итогового пространства.

Соответственно, алгоритмы, не имеющие свойств алгоритмов пространства изображений, относятся к группе алгоритмов, работающих в пространстве объектов.

Для последних лет также стало актуально классифицировать алгоритмы обнаружения столкновений на те, что рассчитаны на работу на CPU, и на те, которые разработаны для GPU. Обусловлено это разделение тем, что потоковая модель вычислений на GPU существенно отличается от модели вычисление на CPU, и, следовательно, ранее разработанные алгоритмы для CPU могут быть переработаны для получения выигрыша от большой вычислительной мощности GPU. В то же время для GPU было разработано немало алгоритмов, не имеющих аналогов для CPU.

3. Обзор алгоритмов узкой фазы

3.1. Алгоритмы для CPU

В данную категорию, прежде всего, попадают алгоритмы на основе иерархий ограничивающих объемов. Листовые узлы подобных иерархий могут содержать либо по одному примитиву объекта (чаще всего это треугольник), либо некоторое подмножество примитивов объекта, в котором в дальнейшем необходимо проверить на пересечение каждую пару. Выбор глубины разбиения влияет на соотношение занимаемой памяти и количества вычислительных ресурсов, затрачиваемых при работе алгоритма.

Для узкой фазы было предложено множество типов ограничивающей геометрии:

- сферы [4];
- выровненные по осям прямоугольные параллелепипеды [11];
- ориентированные прямоугольные параллелепипеды [12, 13];
- дискретно-ориентированные многогранники, или k-DOP (Discrete Orientation Polytope; многогранник, у которого нормали к граням принадлежат заранее определенному фиксированному множеству направлений размерности k) [14] и др.

Выбор ограничивающего объема является компромиссом между вычислительной сложностью проверки на пересечение и тем, насколько хорошо выбранный ограничивающий объем аппроксимирует объект. Примеры ограничивающей геометрии приведены на рисунке 1.

Алгоритм обнаружения столкновений на основе иерархий ограничивающих объемов состоит в одновременном прохождении в ширину или в глубину деревьев, соответствующих двум объектам. На каждом шаге проводится проверка на пересечение ограничивающих объемов. Если ограничивающие объемы текущих узлов не пересекаются, то эти узлы и их поддеревья отбрасываются. Иначе, проводится рекурсивная проверка на пересечение у дочерних узлов, пока не будут достигнуты листовые узлы. На выходе получаются пары листовых

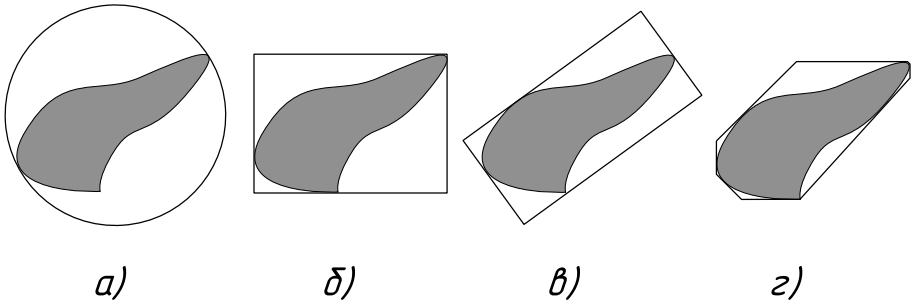


Рис. 1. Примеры ограничивающей геометрии: а) ограничивающая сфера; б) ограничивающий бокс, выровненный по осям (AABB); в) ориентированный ограничивающий бокс (OBB); г) 6-DOP

узлов от каждого из двух объектов, в которых необходимо провести проверку на пересечение на уровне примитивов.

В работе [10] предложено улучшение традиционной схемы путем хранения в листах дерева отдельных элементов (вершин, ребер, граней) так, чтобы каждый элемент присутствовал только в одном листе. Это позволяет значительно сократить количество проверок ребро-ребро, вершина-грань и ребро-грань на последнем шаге алгоритма, когда происходят проверки на уровне отдельных примитивов.

Ввиду того что построение иерархии ограничивающих объемов для объектов сцены является ресурсозатратным процессом, его проводят заранее и ассоциируют с исходными объектами. Таким образом, данный тип алгоритмов в исходном виде хорошо подходит для обнаружения столкновений между твердыми телами. Для деформируемых тел требуется перестроение иерархии или частичное ее обновление, т.к. положение элементов для таких объектов меняется относительно друг друга. Полное перестроение иерархии на каждом временном шаге не может быть использовано для моделирования в реальном времени для даже умеренных по сложности сцен, потому что требует значительных вычислительных затрат. За последние несколько лет в связи с возросшим интересом к моделированию деформируемых тел были разработаны менее ресурсозатратные методы обновления иерархий. Так, в работе [15] был предложен эффективный подход для восьмеричного дерева иерархий описывающих объемов на основе временной когерентности сцены и алгоритма сортировки вставками. Время, затрачиваемое на подобную сортировку, будет тем меньше, чем незначительнее были изменения в положении примитивов объекта. Таким образом, при высокой временной когерентности затраты на обновление иерархии будут малы.

В [16] был предложен подход для ускорения обновления иерархии путем пропуска или упрощения процесса обновления для нескольких временных шагов. Для этой цели описывающий объем расширяют на некоторую величину, и обновление не требуется, пока примитив не сдвинулся более, чем на эту величину. В качестве ограничивающих объемов были использованы 18-DOP'ы.

В качестве базовой структуры данных для представления иерархии ограничивающих объемов для обнаружения столкновений разрушающихся объектов

в работе [17] были использованы АВЛ-деревья. Авторами был представлен алгоритм для динамического изменения структуры дерева в случае изменения позиций, а также при добавлении или удалении отдельных элементов на основе балансировки АВЛ-дерева.

В [18] были использованы динамические деревья ограничивающих объемов. Было предложено проводить обнаружение столкновений в два шага: (1) обновление ограничивающих объемов по всему дереву, (2) запрос на столкновение. Во время первого шага заново просчитываются ограничивающие объемы у существующих узлов, а также помечаются на удаление поддеревья, которые требуют перестройки. Критерием того, следует ли помечать поддерево как негодное для дальнейшего использования, является сравнение отношений объемов старого родительского узла с просчитанным заново. На втором шаге алгоритм обходит деревья объектов в поисках перекрывающихся узлов и перестраивает помеченные на первом шаге поддеревья. Было показано, что такой подход эффективен для обнаружения столкновений между деформируемыми объектами, но для случая самопересечений примитивов внутри одного объекта его эффективность значительно снижается.

Среди алгоритмов узкой фазы можно выделить еще одну группу, в основе которой лежит работа напрямую с вершинами, ребрами и гранями объектов. К этой категории относится алгоритм Лина-Кэнни [19] и V-Clip [20], который устраняет такие недостатки первого, как неспособность работать в случае проникновения объектов друг в друга (а не только касания), неустойчивость в некоторых вырожденных взаимных конфигурациях объектов и сложность реализации.

Популярной группой алгоритмов являются также алгоритмы, основанные на симплексах. Полигональный объект представляется как выпуклая оболочка, и алгоритм работает над симплексами из подмножества множества точек этой оболочки. Наиболее известен алгоритм Гильберта-Джонсона-Кеерти (GJK) [21] и ряд его улучшений. Так, в [22] была добавлена возможность нахождения расстояния между объектами (или глубины проникновения, если они пересекаются). Работа [23] направлена на значительное улучшение стабильности и скорости алгоритма. GJK в явном виде не использует геометрию исходных объектов, а работает над их разностью Минковского. Таким образом, задача нахождения расстояния между двумя объектами сводится к нахождению расстояния от начала координат до разности Минковского объектов.

3.2. Алгоритмы для GPU

Последние 15 лет характеризуются бурным ростом производительности графических адаптеров. Графические адаптеры привлекательны для исследователей в связи с их возможностью работать одновременно с миллионами графических элементов. Также расширение возможностей для программирования с использованием шейдеров приводит к тому, что все большее количество вычислений, не связанных напрямую с выводом графики на экран, переносится с CPU на GPU. Этому способствует и появление унифицированного API OpenCL, обес-

печивающего стандартизированный доступ к видеокарте как к устройству с высокой степенью параллелизма. В настоящее время на рынке доступны потребительские графические адаптеры с низкой ценой и большой производительностью.

3.2.1. Алгоритмы, работающие в пространстве изображений

Для большинства алгоритмов, работающих в пространстве изображений, можно выделить общий ряд достоинств и недостатков. Так, к достоинствам можно отнести то, что часто не требуются шаги для начальной инициализации и обычно не имеет значения, в каком виде поступает геометрия на вход. Недостатком является невозможность получения точного результата из-за ограниченной разрешающей способности пространства изображений и количества бит на пиксель для z-буфера. Также обычным ограничением является требование на замкнутость поверхности у объектов.

Одна из первых работ с использованием методов пространства изображений была представлена в [24]. Метод состоит в отрисовке объектов вдоль заданной оси с сохранением z-значений глубины в списках для каждого пикселя. Отрисовка производится дважды для получения переднего и заднего интервалов объекта. Столкновение может быть обнаружено путем нахождения перекрывающихся интервалов z-значений для объектов. Предложенный метод работает только для выпуклой геометрии.

В [26] было предложено использовать отрисовку объектов в два LDI, по одному на объект, с ограничением видимой области в рамках AABB, соответствующего пересечению двух AABB объектов. Применяя операцию булева пересечения между LDI пары объектов, можно узнать, пересекаются ли объекты. Алгоритм не ограничен только выпуклыми объектами и может быть использован для произвольных объектов с замкнутой поверхностью и для нахождения самопересечений. Авторы показали, что для сложных объектов предложенный метод неэффективен, т.к. требует нескольких проходов рендеринга в LDI и накладных расходов в виде интенсивного чтения из памяти GPU.

Эффективный алгоритм с широким использованием возможностей GPU и OpenGL API, таких как occlusion query, пользовательские буферы кадра и шейдеры, был предложен в [27]. Так же, как и в [26], находится растеризуемый объем в виде AABB, но рендеринг происходит в общий для всех объектов набор текстур с сохранением идентификатора объекта, а не в два LDI. Предполагается, что объекты не самопересекаются и имеют замкнутую поверхность. На выходе алгоритма получаются точки столкновения между объектами, и число объектов, участвующих в столкновении, может быть больше двух.

Схожий с предыдущим подход, но на основе возможностей DirectX 10, был предложен в [28]. Рендеринг производится одновременно в восемь слоев глубины, таким образом значительно сокращая количество вызовов графического API и повышая общую производительность. Уменьшение потока GPU $\rightarrow$ CPU было достигнуто благодаря использованию геометрического шейдера, позволившего отбрасывать не несущие информацию о столкновении элементы. В каче-

стве результата алгоритм возвращает пары треугольников исходных объектов, которые требуется проверить на пересечение на CPU.

Еще один подход на основе LDI представлен в [3]. Область возможного пересечения отрисовывается в три LDI, по одному для каждой из взаимно-перпендикулярных осей видимости. Далее определяется объем, занимаемый пересечением объектов:

$$V = a \sum_{(i,j) \in C} (-1)^d z_{ij},$$

где a — площадь одного пиксела, z — глубина, d — 1 или 2 в зависимости от порядка пиксела в LDI, C — множество индексов фрагментов, покрывающих поверхность объема пересечения. Далее, используя объем пересечения, авторы предлагают получить на его основе величину расталкивающей силы для столкнувшихся объектов. Основной недостаток алгоритма состоит в использовании трех LDI и соответственно трех проходов рендеринга: это снижает производительность из-за большей нагрузки на видеокарту и увеличивает объем занимаемой памяти.

Принцип, лежащий в работе алгоритмов [3, 24, 26–28], достаточно прост. Два луча, выпущенных вдоль оси Z через столкнувшиеся объекты, пересекают их и образуют интервалы $[z_{A0}, z_{A1}], [z_{B0}, z_{B1}]$ и $[z'_{A0}, z'_{A1}], [z'_{B0}, z'_{B1}]$ для первого и второго лучей соответственно, как показано на рисунке 2, а. При этом столкновение можно обнаружить, анализируя на перекрытие полученные интервалы для оси Z, т.е. значения из буфера глубины графического адаптера после рендеринга. Для повышения точности алгоритма и производительности рендеринг выполняется для пересечения двух AABB объектов. Результирующий AABB называется *регион интереса* или *объем интереса*. С помощью графического API выставляется ортографическая проекция с направлением вида по одной из осей координат с ограничением области видимости по полученному AABB (закрашен серым цветом на рисунке 2, б).

Попытка решить проблему низкой пропускной способности шины GPU → CPU, как основной причины низкой производительности алгоритмов на GPU, была предпринята в пакете для обнаружения столкновений CULLIDE [25]. Предложенный алгоритм проверяет видимость объекта O относительно множества объектов S. Множество потенциально пересекающихся примитивов определяется с помощью запроса к видеокарте, загорожен ли O объектами из S. Если не загорожен, то O не пересекается с S и, следовательно, не включается в множество потенциально пересекающихся примитивов. CULLIDE делит исходный объект на множество подобъектов на предварительной стадии и далее проверяет видимость путем прохода по иерархии подобъектов. Недостатком данного алгоритма является дорогостоящая процедура на предварительном шаге, что ограничивает использование этого подхода для разрушаемых тел.

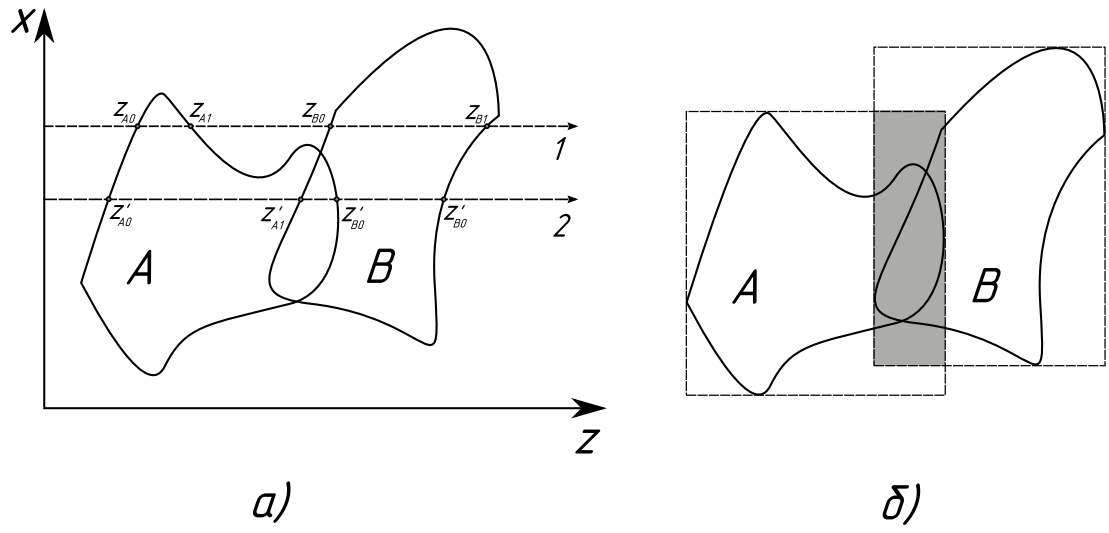


Рис. 2. Принцип работы большинства алгоритмов в пространстве изображений: а) скан-линии 1 и 2 вдоль оси Z для двух пересекающихся объектов; б) нахождения региона интереса

3.2.2. Алгоритмы, работающие в пространстве объектов

Появление программируемости в широко распространенных потребительских графических адаптерах и быстрое наращивание их вычислительной мощности явилось толчком для исследований в направлении разработки алгоритмов обнаружения столкновений не только в рамках пространства изображений, но и в пространстве объектов, что позволило бы избавиться от недостатков алгоритмов пространства изображений, перечисленных в предыдущем подразделе.

Одним из первых алгоритмов в пространстве объектов на GPU является [29]. Подход основан на использовании балансированных иерархий ограничивающих объемов, обход деревьев осуществляется в ширину, вместо обычно используемого обхода в глубину. Все вычисления во время обхода дерева, включая проверки на пересечения между примитивами, реализованы на фрагментных и вершинных шейдерах. Алгоритм не имеет ограничений на форму, топологию или замкнутость поверхности объектов, но может быть использован только для твердых тел. Авторы отмечают, что алгоритм лишь немногим быстрее версии, реализованной для CPU.

В работе [30] предложен подход на основе дерева из AABB для обнаружения столкновений между деформируемыми объектами, заданными NURBS кривыми. Описаны процедуры построения деревьев ограничивающих объемов на GPU в реальном времени, эффективная техника обхода дерева с одновременным сокращением потока, поступающего на CPU (stream reduction) на основе мипмап-уровней текстур, а также дополнение в виде алгоритма для определения самопересечения. Было показано, что для объектов умеренного размера реализация алгоритма работает в реальном времени.

В [31] дерево из AABB для каждого из объектов передается в GPU как

два списка, или потока. В представлении графического API эти списки представлены как 1D текстуры, где каждый элемент такой текстуры в R, G и B компонентах хранит координаты углов AABB, а в альфа-компоненте признак наличия этого узла в дереве, т.к. передаются только листовые узлы самого нижнего уровня дерева, и не все они могут присутствовать в дереве для объекта. Далее на GPU выполняются всевозможные попарные проверки между двумя потоками для двух объектов, а результат для каждого элемента из дерева записывается как ложь/истина и передается для дальнейшей проверки на CPU на предмет пересечения соответствующих узлам треугольников. Для деформируемых тел иерархия каждый раз обновляется. В результате при реализации на шейдерах получается достаточно высокая производительность, авторы отмечают трехкратное превосходство по скорости над CULLIDE [25], который был описан в предыдущем подразделе.

4. Заключение

За последние три десятилетия в литературе было представлено большое количество методов и алгоритмов для обнаружения столкновений. Каждый из них имеет недостатки или может быть использован только при ограничениях на способ задания формы объектов, характер движения объектов или на тип аппаратного обеспечения, используемого для моделирования сцены и т.п. Тем не менее на данный момент наиболее широко используются алгоритмы на основе иерархий ограничивающих объемов, а также постепенно увеличивается доля алгоритмов, созданных для работы на параллельных потоковых программируемых устройствах, к которым относятся современные GPU.

Литература

1. Ericson C. Real-Time Collision Detection. San Franciso: Morgan Kaufmann Publishers, 2005. 593 p.
2. Bergen G. Collision Detection in Interactive 3D Environments. San Franciso: Morgan Kaufmann Publishers, 2004. 278 p.
3. Faure F. Image-based Collision Detection and Response between Arbitrary Volume Objects / F. Faure, S. Barbier, J. Allard, F. Falipou // Proceedings of the 2008 ACM SIGGRAPH Eurographics Symposium on Computer Animation. 2008. P. 155–162.
4. Hubbard P. Collision Detection for Interactive Graphics Applications // IEEE Transactions on Visualization and Computer Graphics. 1995. V. 1, N. 3. P. 218–230.
5. Baraff D. Dynamic Simulation of Non-Penetrating Rigid Bodies: PhD thesis. Cornell University, 1992.
6. Tracy D., Buss S., Woods B. Efficient Large-Scale Sweep and Prune Methods with AABB Insertion and Removal // Proceedings of the 2009 IEEE Virtual Reality Conference. 2009. P. 191–198.
7. Le Grand S. Broad-Phase Collision Detection with CUDA // GPU Gems 3 / edt. by Hubert Nguyen. 2007. P. 697–723.

8. Luque R., Comba J., Freitas C. Broad-Phase Collision Detection Using Semi-Adjusting BSP-Trees // Proceedings of the symposium on Interactive 3D graphics and games. 2005. P. 179–186.
9. Layered Depth Images / J. Shade [and others] // Proceedings of the 25th conference on Computer graphics and interactive techniques. New York, 1998. P. 231–242.
10. Curtis S., Tamstorf R., D. Manocha D. Fast Collision Detection for Deformable Models using Representative-Triangles // Proceedings of the 2008 symposium on Interactive 3D graphics and games. 2008. P. 61–69.
11. van den Bergen G. Efficient Collision Detection of Complex Deformable Models Using AABB Trees // Journal of Graphics Tools. 1997. V. 2, N. 4. P. 1—13.
12. Zachmann G., Felger W. The BoxTree: Enable Real-time and Exact Collision Detection of Arbitrary Polyhedra // Proceedings of SIVE'95. 1995. P. 104–113.
13. Gottschalk S., Lin M.C., Manocha D. OBBTree: a hierarchical structure for rapid interference detection // Proceedings of Computer graphics and interactive techniques. 1996. P. 171–180.
14. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs / J. Klosowski [and others] // IEEE Transactions on Visualization And Computer Graphics. 1998. V. 4, N. 1. P. 21–36.
15. Ganovelli F., Dingliana J., O'Sullivan C. BucketTree: Improving Collision Detection Between Deformable Objects // Proceedings of SCCG2000 Spring Conf. on Comp. Graphics. 2000. P. 156–163.
16. Mezger J., Kimmerle S., Etzmuss O. Hierarchical Techniques in Collision Detection for Cloth Animation // Journal of WSCG. 2003. V. 11, N. 2. P. 322–329.
17. Balanced Hierarchies for Collision Detection between Fracturing Objects / M.A. Otaduy [and others] // Proceedings of 2007 IEEE Virtual Reality Conference. 2007. P. 83–90.
18. Larsson T., Akenin-Moeller T. A Dynamic Bounding Volume Hierarchy for Generalized Collision Detection // Computer and Graphics. 2006. V. 30, N. 3. P. 451–460.
19. Lin M. C. Efficient Collision Detection for Animation and Robotics: PhD thesis. Berkeley: University of California, 1993. 147 p.
20. Mirtich B. V-Clip: Fast and Robust Polyhedral Collision Detection // ACM Transactions on Graphics. 1998. V. 17, N. 3. P. 177–208.
21. Gilbert E.G., Johnson D.W., Keerthi S.S. A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space // IEEE Journal of Robotics and Automation. 1998. V. 4, N 2. P. 193–203.
22. Cameron S. Enhancing GJK: Computing Minimum and Penetration Distance between Convex Polyhedra // Proceedings IEEE Int. Conf. on Robotics and Automation. 1997. P. 3112–3117.
23. van den Bergen G. A Fast and Robust GJK Implementation for Collision Detection of Convex Objects // Journal of Graphics Tools. 1999. V. 4, N. 2. P. 7–25.
24. Shinya M., Forgue M.-C. Interference Detection Through Rasterization // The Journal of Visualization and Computer Animation. 1991. V. 2, N. 4. P. 132–134.
25. Govindaraju N., Lin M.C., Manocha D. Quick-CULLIDE: Fast Inter- and Intra-object Collision Culling Using Graphics Hardware // Proceedings of IEEE Virtual Reality. 2005. P. 59–66.
26. Heidelberger B., Teschner M., Gross M. Detection of Collisions and Self-collisions Using

Image-space Techniques // Journal of WSCG. 2004. V. 12, N. 3. P. 145–152.

27. Jang H.-Y., Jeong T.S., Han J.H. GPU-based Image-space Approach to Collision Detection among Closed Objects. 2006. URL: http://drjang.kr/Formal/paper/2006_GPU-based_Imate-space_Approach_to_Collision_Detection_among_Closed_Objects_PG.pdf (01.03.2010).
28. Jang H.-Y., Han J.H. Fast collision detection using the A-buffer // Visual Computer. 2008. V. 24, N. 7-9. P. 659–667.
29. Gress A., Zachmann G. Object-space Interference Detection on Programmable Graphics Hardware // Proceedings of SIAM Conf. Geometric Design and Computing. 2004. P. 311–328.
30. Gress A., Guthe M., Clein R. GPU-based Collision Detection for Deformable Parameterized Surfaces // Proceedings of Eurographics 2006. V. 25, N. 3. P. 497–506.
31. Zhang X., Kim Y.J. Interactive Collision Detection for Deformable Models using Streaming AABBs // IEEE Transactions on Visualization And Computer Graphics. 2007. V. 13, N. 2. P. 31–329.

Математические структуры и моделирование
2010, вып. 21, с. 96–103

УДК 004.925

ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ СВЯЗЫВАНИЯ CUDA С ГРАФИЧЕСКИМИ API НА ПРИМЕРЕ ЗАДАЧИ SAXPY

А.Ю. Суравикин, В.В. Коробицын

В статье исследована CUDA как одна из технологий GPGPU. Создана программа решения задачи SAXPY на CUDA, описаны способы организации взаимодействия с графическими API. Предоставлены листинги базовой программы и функций обмена данных с OpenGL. Приведено сравнение времени расчетов с аналогичными программами, которые используют другие технологии GPGPU.

Введение

На сегодняшний день существует множество задач, которые требуют высоких вычислительных затрат и имеют параллельные алгоритмы решения. При этом современные графические процессоры (GPU — Graphics Processing Units) обладают параллельной архитектурой и высокой производительностью, поэтому могут эффективно применяться для решения подобных задач. Однако достигнуть высокой скорости решения задач можно лишь после значительной оптимизации алгоритма под параллельную архитектуру и конкретный GPU. Научное направление по созданию, реализации и оптимизации различных алгоритмов, напрямую не связанных с компьютерной графикой, получило названием GPGPU (General Purpose computations on Graphics Processing Units — вычисления общего назначения на графических процессорах).

Компания NVIDIA предложила свое решение, предназначенное для разработки приложений для массивно-параллельных вычислительных устройств, и назвала его CUDA (Compute Unified Device Architecture — унифицированная архитектура компьютерных вычислений). CUDA ориентирована на графические процессоры NVIDIA GeForce 8-й серии и новее, а также специализированные процессоры NVIDIA Tesla. Технология активно развивается и поддерживается разработчиками ПО. Для CUDA разработаны вспомогательные библиотеки: CUBLAS, CUDA Performance Primitives, thrust. В состав пакета CUDA Toolkit входит компилятор PTX (Parallel Thread eXecution), позволяющий работать с ассемблерным кодом программ, выполняемым на графическом процессоре.

Омский государственный университет им. Ф.М. Достоевского.
E-mail: tizona_del_cid@rambler.ru

Целью настоящей статьи является описание подхода создания несложной программы на CUDA и способов обмена данными с графическими API. Для этого реализовано решение простейшей задачи линейной алгебры SAXPY (Scalar Alpha X Plus Y) — задачи скалярного умножения и векторного сложения [1]. Она заключается в вычислении результата двух векторных операций: умножения на скаляр и сложения векторов. Необходимо вычислить новое значение вектора $\mathbf{y}$ по формуле

$$\mathbf{y} = \alpha \cdot \mathbf{x} + \mathbf{y},$$

где α — число, $\mathbf{x}$ и $\mathbf{y}$ — векторы большой размерности.

1. Описание технологии CUDA

Программы для CUDA (соответствующие файлы обычно имеют расширение .cu) пишутся на расширении языка C и компилируются при помощи компилятора nvcc, предоставляемого бесплатно компанией NVIDIA через Интернет.

Перечислим основные расширения языка C, которые введены в CUDA:

1) спецификаторы функций, показывающих, где будет выполняться функция (host или device) и откуда она может быть вызвана;

2) спецификаторы переменных, задающие тип памяти, используемый для данной переменной;

3) директива, предназначенная для запуска ядра программы на GPU, задающая как данные, так и иерархию потоков выполнения;

4) встроенные переменные, содержащие информацию о выполняемом потоке;

5) библиотека, включающая в себя дополнительные типы данных и функции работы с памятью, управления устройством и другое.

Описание архитектуры CUDA и пример использования можно посмотреть в [2], [3].

Функция решения задачи SAXPY на CUDA входит в библиотеку CUBLAS [4], [5]. Однако мы представляем собственную реализацию для оценки возможностей разработки с использованием инструмента CUDA.

2. Пример программы CUDA

Ниже представлен код программы CUDA, который осуществляет инициализацию и загрузку данных на вычислительное устройство (device). Такой код выполняется в модуле, компилируемом программой nvcc, однако некоторыми функциями CUDA можно воспользоваться и в модуле C++ с подключенными заголовочными файлами и библиотеками CUDA.

```
   cudaSetDevice(cutGetMaxGflopsDeviceId());
   // параметры данных
   vectorSize = n;
   dataSize = n * sizeof(float);
   // выделим память для векторов на стороне GPU
   cudaMalloc(&vectorX, dataSize);
   cudaMalloc(&vectorY, dataSize);
   // ... и на CPU
   cudaMallocHost(&vectorXHost, dataSize);
   cudaMallocHost(&resultHost, dataSize);
   // инициализация входных данных:
   // заполним массив памяти CPU (Host) псеводслучайными числами
   // в интервале [0; 1]
   srand(timeGetTime());
   for (int i = 0; i < vectorSize; i++)
   {
      vectorXHost[i] = (float)rand() / RAND_MAX;
   }
   // скопируем данные массива в вектор X на GPU (Device)
   cudaMemcpy(vectorX, vectorXHost, dataSize,
              cudaMemcpyHostToDevice);
   // вектор Y обнулим
   cudaMemset(vectorY, 0, dataSize);
```

Далее представлена функция обработки данных, в которой устанавливаются размеры блока выполнения ядер (kernel) [7]), запускается ядро и копируется результат вычислений в память CPU.

```
void    ProcessData(int iterations)
{
   // размер блока - некоторая константа
   dim3 block(blockSize);
   // количество блоков зависит от общего числа данных
   dim3 grid(vectorSize / block.x);
   // производим несколько итераций запуска ядра
   for (int i = 0; i < iterations; i++)
   {
      // передаем параметр \alpha и указатели на векторы X и Y
      kernel_SAXPY<<<grid, block>>>(0.5f, vectorX, vectorY);
   }
   // копируем обработанные данные в host-память
   cudaMemcpy(resultHost, vectorY, dataSize, cudaMemcpyDeviceToHost);
}
```

Обратим внимание на размеры блоков выполнения. Их значения необходимо подбирать для каждого ядра таким образом, чтобы максимально загрузить

устройство (графический процессор). Для этого оцениваем число занимаемых ядром регистров. Чем меньше регистров используется в ядре, тем больше блоков можно запустить одновременно. Теперь рассмотрим программу-ядро, выполняемую на GPU:

```
__global__ void kernel_SAXPY(float alpha, float* X, float* Y)
{
   uint addr = __umul24(blockIdx.x, blockDim.x) + threadIdx.x;
   Y[addr] = alpha * X[addr] + Y[addr];
}
```

В приведенном листинге ключевое слово __global__ определяет функцию как ядро. В самой функции сначала рассчитываем адрес обрабатываемого элемента, исходя из номера потока, номера и размера блока. Для целочисленного умножения небольших чисел используется функция __umul24(), которая выполняется значительно быстрее, чем полноценное умножение 32-битных чисел. Далее совершаются собственно операции SAXPY: каждый поток вычисляет один элемент вектора, происходит чтение из элементов векторов X, Y и запись результата в элемент вектора Y.

После обработки и копирования данных в память CPU их можно вывести любым способом, доступным для языка C++. Удаление указателей в памяти устройства осуществляется функцией cudaFree(), в памяти хоста — функцией cudaFreeHost().

3. Взаимодействие CUDA с графическими API

Во многих случаях результаты расчетов требуется вывести в графическом виде. Для этого используются специальные функции взаимодействия с графическим API.

3.1. Взаимодействие с OpenGL

В CUDA версии 2.3 взаимодействие с OpenGL ограничено обменом данных с OpenGL Buffer Objects, т.е. с объектами-буферами [7]). Рассмотрим процесс обмена данных между памятью CUDA и буферами OpenGL.

Для работы с OpenGL требуется, чтобы устройство CUDA было указано функцией cudaGLSetGLDevice() перед всеми вызовами функций CUDA. Прежде чем проецировать буфер для CUDA, необходимо его зарегистрировать следующей функцией:

```
GLuint bufferObj;
cudaGLRegisterBufferObject(bufferObj);
```

Когда буфер зарегистрирован, ядро может считывать из него данные или записывать в него. Для обращения необходимо использовать адрес, возвращаемый функцией cudaGLMapBufferObject().

```
GLuint bufferObj;
float* vectorYPtr;
cudaGLMapBufferObject((void**)&vectorYPtr, bufferObj);
```

Например, можно выполнить следующий код:

```
kernel_SAXPY<<<grid, block>>>(0.5f, vectorX, vectorYPtr);
```

После выполнения функции ядра в буфере bufferObj будет содержаться вектор Y, который можно использовать для рендеринга, например в качестве координат точек, трансформируемых далее в вершинном шейдере.

Отключение проекции и регистрации осуществляется при помощи функций cudaGLUnmapBufferObject()и cudaGLUnregisterBufferObject(). При обработке изображений или других двумерных массивов данных имеет смысл передавать не вершинную информацию, а пиксельную. Для этого используется объект Pixel Buffer Object (PBO), который создается следующим образом:

```
glGenBuffers(1, &pboId);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pboId);
glBufferData(GL_PIXEL_UNPACK_BUFFER, dataSize, 0, GL_DYNAMIC_DRAW);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER, 0);
```

Здесь pboId – идентификатор буфера-PBO, dataSize – его размер. От создания вершинного буфера он отличается константой GL_PIXEL_UNPACK_BUFFER, обозначающей чтение (распаковку — unpack) командами вида glDrawPixels и glTexImage2D пиксельных данных из буфера. О расширении библиотеки OpenGL ARB_pixel_buffer_object подробнее изложено в [8]. О буферах PBO как части стандарта OpenGL 3.2 в [9]. Буфер pboId один раз регистрируется в CUDA, затем каждый кадр отрисовки проецируется для запуска ядра, далее ядро выполняет расчеты, после чего проекция закрывается. Для рендера полученного изображения копируем данные из PBO в текстуру texId. Следующий код иллюстрирует процесс копирования:

```
glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pboId);
glBindTexture(GL_TEXTURE_2D, texId);
glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, width, height, GL_RGBA,
                GL_FLOAT, 0);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER, 0);
```

Теперь текстуру texId можно наложить на прямоугольник и произвести ее рендер любым подходящим способом. Отметим также, что в бета-версии CUDA 3.0 стало доступным взаимодействие с текстурами OpenGL напрямую.

3.2. Взаимодействие с Direct3D

Взаимодействие CUDA с Direct3D описано в [7]. Приведем лишь основные моменты реализации.

Прежде чем использовать вызовы библиотеки, необходимо передать устройство Direct3D 9 в функцию cudaD3D9SetDirect3DDevice() (для Direct3D 10, соответственно, cudaD3D10SetDirect3DDevice()). Ресурсы Direct3D 9 (вершинные буферы IDirect3DVertexBuffer9, поверхности IDirect3DSurface9 и т.д.) и Direct3D 10 (буферы ID3D10Buffer, и текстуры ID3D10Texture2D и т.д.) ассоциируются функциями cudaD3D9RegisterResource(), cudaD3D10RegisterResource(). Для работы с ресурсами нужно спроецировать их память. Проекция массива ресурсов открывается функцией cudaD3D9MapResources() и закрывается функцией cudaD3D9UnmapResources().

В бета-версии CUDA 3.0 добавлена возможность взаимодействия с Direct3D версии 11.

4. Компьютерный эксперимент

Для оценки производительности реализации на CUDA был проведен компьютерный эксперимент. Определялось время выполнения задачи SAXPY на CUDA и других технологиях GPGPU: DirectCompute и OpenCL. Измерения производились следующим образом: сначала подготавливаются все данные для ядра, затем запускается таймер, после чего выполняется функция-ядро. Далее выполняется синхронизация с CPU любым доступным способом, и таймером замеряется результат. Конфигурация тестовой системы: процессор – Core 2 Duo 2.26 ГГц, оперативная память объемом 4ГБ, видеокарта на основе видеопроцессора GeForce 9800M GTS с 1ГБ видеопамяти.

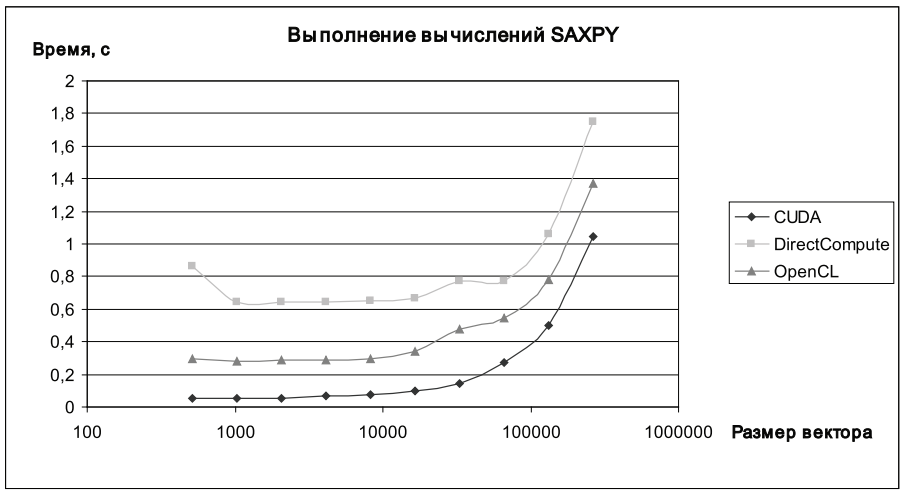


Рис. 1. Сравнение времени вычисления SAXPY на различных API

Графики на рисунке 1 показывают время выполнения операции SAXPY различными API на векторах размером от 512 до 262144 элементов. Видно, что CUDA демонстрирует самую высокую скорость выполнения. OpenCL и DirectCompute отстают по скорости. Возможно, это связано с лишними затратами времени на установку состояния API и синхронизацию с GPU при запус-

ке каждой итерации расчета. Вызовы функций OpenCL и DirectCompute осуществляются через CUDA, поэтому на выполнение требуется дополнительное время.

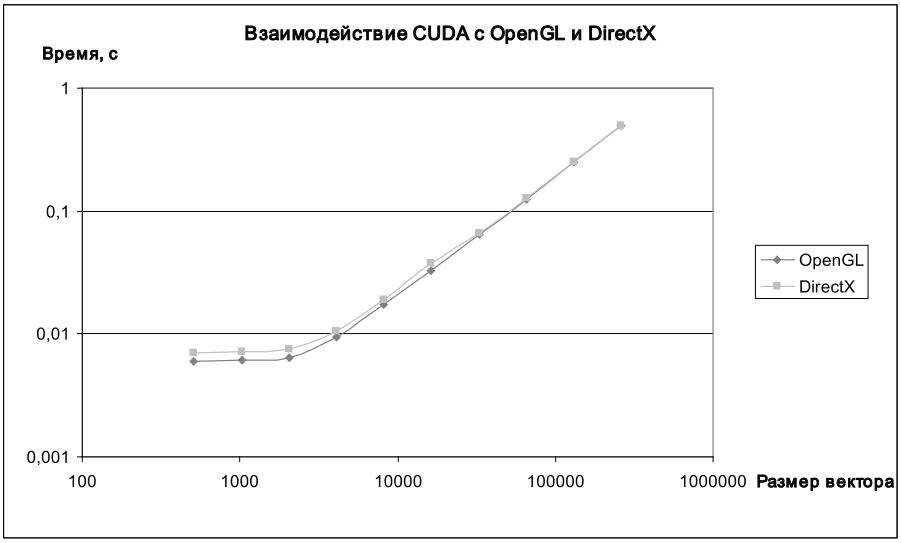


Рис. 2. Время расчета SAXPY на CUDA и вывода результатов с помощью графических API

Второй эксперимент заключался в расчете SAXPY и выводе результатов на экран с помощью средств графических API, т.е. в данном случае проводилось сравнение производительности работы CUDA с OpenGL 3.0 и DirectX 11. Обмен данными производился с помощью копирования данных из буфера. График на рисунке 2 показывает, что при малых объемах данных OpenGL работает быстрее, но при увеличении размера вектора различия становятся минимальными.

5. Выводы

С использованием технологии CUDA было реализовано решение задачи линейной алгебры SAXPY. Также получена программа, которая пересылает исходные данные на GPU, обрабатывает их и возвращает результат. После этого рассмотрено взаимодействие CUDA с графическими API. В случае OpenGL был описан способ обмена данными через Pixel Buffer Object. Для проведения экспериментов также были реализованы программы расчета SAXPY на CUDA, OpenCL, DirectCompute и вывод результатов в графическом виде с использованием OpenGL 3.0 и DirectX 11. Мы не ставили своей задачей освоить работу с разделяемой памятью, хотя оптимизация вычислений под разделяемую, локальную, константную память позволяет значительно повысить производительность приложений CUDA. Это является серьезным преимуществом по сравнению с использованием графического API, так как позволяет оптимизировать программу под конкретную архитектуру оборудования.

Сравнение времени выполнения реализаций алгоритма на CUDA, OpenCL и DirectCompute показало, что взаимодействие с дополнительными API (т.е.

выполнение функций OpenCL и DirectCompute через CUDA) добавляет значительные задержки, особенно при большом числе итераций и незначительной вычислительной нагрузке. Следовательно, для ядер с большими вычислительными затратами API не будет иметь значения, а для большого количества ядер с меньшим временем выполнения CUDA будет выполняться быстрее. Эксперимент с выводом результатов вычислений из CUDA на экран с помощью графических API показал, что производительность одинакова при использовании как OpenGL, так и DirectX.

Литература

1. Описание алгебраической функции SAXPY // Википедия. URL: http://en.wikipedia.org/wiki/SAXPY (дата обращения: 28.02.2010).
2. Боресков А. Основы CUDA. URL: http://steps3d.narod.ru/tutorials/cuda-tutorial.html (дата обращения: 28.02.2010).
3. Zibula A. General Purpose Computation on Graphics Processing Units (GPGPU) using CUDA // Parallel Programming and Parallel Algorithms Seminar. Munster: Westfalische Wilhelms-Universitat, 2009.
4. Библиотека CUBLAS для CUDA. URL: http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/ docs/CUBLAS_Library_2.3.pdf (дата обращения: 28.02.2010).
5. Страницы загрузок средств разработки CUDA 2.3. URL: http://developer.nvidia.com/object/cuda_2_3_downloads.html (дата обращения: 28.02.2010).
6. CUDA 3.0 beta. URL: http://forums.nvidia.com/index.php?showtopic=149959 (дата обращения: 28.02.2010).
7. NVIDIA CUDA Programming Guide. URL: http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/ NVIDIA_CUDA_Programming_Guide_2.3.pdf (дата обращения: 28.02.2010).
8. Спецификация расширения OpenGL ARB_pixel_buffer_object. URL: http://www.opengl.org/registry/specs/ARB/pixel_buffer_object.txt (дата обращения: 28.02.2010).
9. Спецификация OpenGL 3.2. URL: http://www.opengl.org/registry/doc/glspec32.core.20091207.pdf (дата обращения: 28.02.2010).

Математические структуры и моделирование
2010, вып. 21, с. 104–112

УДК 004.056.0

РЕАЛИЗАЦИЯ СИСТЕМЫ ОБНАРУЖЕНИЯ ВТОРЖЕНИЙ КАК ЧАСТИ ИСКУССТВЕННОЙ ИММУННОЙ СИСТЕМЫ

М.Ю. Ваганов

В статье рассматривается реализация системы обнаружения вредоносных программ на основе анализа активности рабочей станции.

Введение

Современный мир невозможно представить без средств коммуникаций и вычислительной техники, в которых главенствующую роль играет программное обеспечение. Информационные технологии прогрессируют очень быстро, охватывая все более широкие области человеческой деятельности: информатизация в отраслях производства, бизнеса, образования, государственных учреждениях и т.д., интеграция автоматизированных систем в локальные и глобальные вычислительные сети. Поэтому безопасность информационных технологий является одним из важнейших аспектов обеспечения их функционирования.

Постоянно растущие масштабы распространения различных компьютерных угроз, основной целью которых, часто является хищение конфиденциальных данных, заставляют разработчиков антивирусного программного обеспечения искать новые, проактивные методы в борьбе с неизвестными угрозами. Одним из таких направлений является использование моделей и решений, уже достаточно хорошо проработанных в эпидемиологии и иммунологии [1].

Целью данной работы является реализации системы обнаружения вредоносных программ на основе анализа активности рабочей станции, являющейся первым шагом в создании модели искусственной иммунной системы, способной реагировать как на известные, так и на новые, не встречавшиеся ранее разновидности вредоносного программного обеспечения.

1. Иммунные системы

Иммунная система живого организма – сложный комплекс структур, способный различать нормальную и аномальную жизнедеятельность клеток, которая может быть представлена как результат действия различных возбудителей, в

Омский государственный университет им. Ф.М. Достоевского.
E-mail: asterxpst@gmail.com

том числе бактерий и вирусов. Конечной целью иммунной системы является формирование иммунного ответа, ответственного за уничтожение чужеродного агента, которым может оказаться болезнетворный микроорганизм, инородное тело, ядовитое вещество или переродившаяся клетка самого организма.

Все формы иммунного ответа можно разделить на приобретенные и врожденные. Основное различие между ними состоит в том, что приобретенный иммунитет высокоспецифичен по отношению к конкретному типу антигенов и позволяет быстрее и эффективнее уничтожать их при повторном столкновении.

Ввиду непрекращающейся адаптации возбудителей, изменения методов распространения и инфицирования также не останавливается эволюционное развитие иммунных систем, в результате которого последние приобрели такие важные особенности, как адаптируемость, наличие памяти, толерантность (терпимость, по отношению к аномалиям, произошедшим на ранних стадиях функционирования системы) и избирательность.

2. Искусственная иммунная система

Под искусственной иммунной системой принято понимать программные комплексы, принципы функционирования которых аналогичны иммунным системам живых организмов [2]. В данной работе предпринимается попытка создания программного комплекса, использующего принципы иммунной системы [3] для детектирования вредоносного кода. В качестве целевой платформы выбраны операционные системы семейства Windows, а именно: XP, 2000, Server 2003, Vista, Server 2008 и 7. В качестве критериев состояния системы выбраны статистические параметры, на которые в большинстве случаев оказывают влияние вредоносные программы. Все параметры разбиты на четыре группы:

- Связанные с работой файловой системы (создание/модификация исполняемых файлов, создание файлов с нестандартными именами, создание файлов с расширениями, не соответствующими заголовку и т.п.).
- Связанные с реестром операционной системы (модификация и удаление ключей, в том числе ключей, гарантированно не связанных с наблюдаемым процессом).
- Связанные с сетью (обращение к устройству \\DEVICE\\TCP и т.п.)
- Связанные с запуском и работой других процессов (запуск процессов с уровнем доступа, допускающим его остановку или запись в его адресное пространство, доступ к \\DEVICE\\PHYSICALMEMORY, открытие потоков других процессов, использование хуков, выгрузка системных процессов).

Каждому параметру приписывается вес, характеризующий степень его опасности. Значения весовых коэффициентов определялись в процессе компьютерного эксперимента. Для инициализации системы выбирается гарантированно не зараженная система в виде вектора параметров. В процессе работы системы

подсистема защиты набирает статистику параметров и периодически вычисляет отклонения состояния системы от «здорового». При обнаружении значительных отклонений идентифицируется подозрительное приложение и переводится в карантин. В дальнейшем приложения, попавшие в карантин, проверяются антивирусными средствами.

2.1. Структура системы

Рассмотрим общую структуру системы, алгоритмы работы отдельных модулей, а так же характеристики приложений, использующиеся в процессе выявления подозрительной или вредоносной активности.

Программный комплекс состоит из ядра и нескольких групп модулей, работающих в пользовательском режиме и в режиме ядра с различным уровнем привилегий (рис. 1).

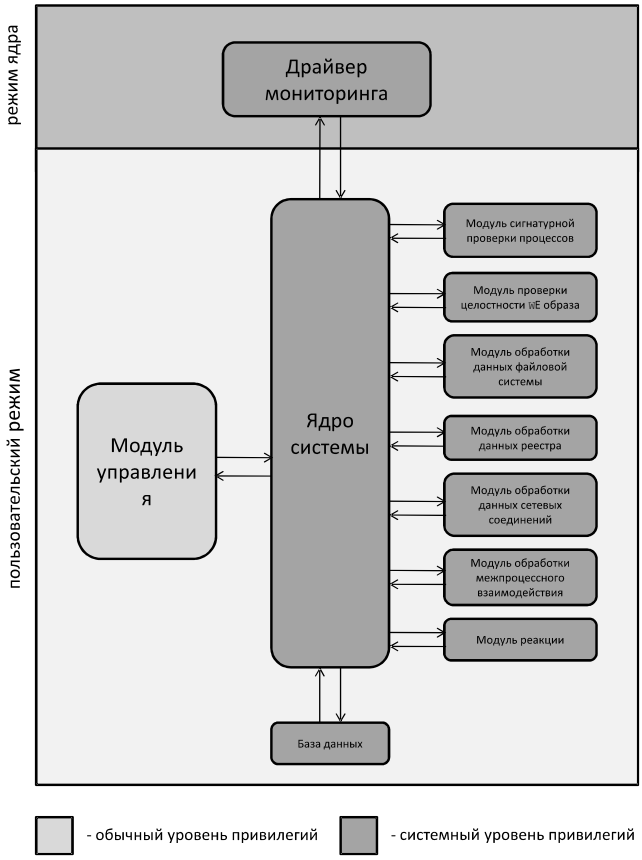


Рис. 1. Структура системы

Рассмотрим более подробно каждый из модулей.

- Ядро. Представляет собой Windows Service, запускаемый с привилегиями SYSTEM в процессе загрузки операционной системы. Основной задачей,

возложенной на ядро, является обеспечение коммуникации между драйвером мониторинга и модулями обработки информации. Так же осуществляется взаимодействие с GUI-модулем управления, позволяющее изменять настройки системы в процессе работы.

- Драйвер мониторинга. Представляет собой WDM Kernel mode driver, включающий функции сбора различных проявлений активности процессов, а именно: работу с файловой системой и устройствами, сетевую и межпроцессную активность. Также отвечает за создание и обновление таблицы процессов, которая требуется для выявления скрытых и замаскированных процессов. В целях повышения производительности и стабильности работы системы, а так же упрощения отладки на стадии разработки обработка собранной информации осуществляется отдельными модулями, функционирующими в пользовательском режиме.

- Модули обработки данных. Занимаются сортировкой информации, получаемой от драйвера мониторинга. Именно здесь принимается решение о том, является ли активность приложения или процесса подозрительной.

- GUI-модуль управления. Является Windows Forms приложением, позволяющим пользователю или администратору производить тонкую настройку системы. К ключевым возможностям относятся:

  – загрузка и выгрузка модулей;

  – изменение чувствительности системы;

  – возможность создавать правила и исключения обработки активности для отдельных процессов и групп процессов.

  Связь модуля управления и ядра системы осуществляется посредством зашифрованного TCP/IP соединения, поэтому настройка может производиться как локально, так и удаленно.

- Модуль реакции. Отвечает за остановку работы процесса в случае признания деятельности последнего вредоносной.

- База данных. Хранит статистическую информацию о деятельности всех процессов системы.

Все компоненты, за исключением драйвера мониторинга, реализованы на языке C# с использованием .NET Framework версии 4.0. Для написания драйвера использовалась комбинация языков C++ и Assembler.

2.2. Процесс загрузки системы

Рассмотрим пошаговую загрузку системы:

1. Windows загружает драйвер мониторинга.

2. Драйвер мониторинга загружает ядро системы и приступает к процедуре инициализации, включающей в себя следующие шаги:

   - составление таблицы процессов с использованием разных методик;
   - установка обработчиков на всевозможные системные события (I/O события, создание сокет-соединений и т.п.).

3. Ядро системы приступает к процедуре инициализации, включающей в себя:

   - проверка целостности модулей путем сверки контрольных сумм;
   - чтение настроек из конфигурационных файлов;
   - загрузку модулей.

2.3. Построение таблицы процессов

Одной из важных характеристик процесса является его видимость для пользователя в диспетчере задач. Практически для всех процессов, за исключением нескольких системных, скрытость воспринимается как негативная характеристика. Для эффективного выявления скрытых процессов нам требуется составление собственной версии таблицы процессов. Ввиду достаточно большого количества вариантов сокрытия процесса в данной работе одновременно используется несколько различных способов получения списка процессов, реализованных как в пользовательском режиме, так и в режиме ядра.

В пользовательском режиме использовались следующие способы получения списка процессов:

- ToolHelp API (вызов функции CreateToolhelp32Snapshot);
- Native API (вызов функции ZwQuerySystemInformation);
- использование списка открытых хендлов;
- для процессов, имеющих окна, можно использовать GetWindowThreadPro-cessId;
- использование прямого системного вызова.

В режиме ядра использовались следующие способы получения списка процессов:

- использование ZwQuerySystemInformation;
- анализ данных структуры EPROCESS;
- анализ списков потоков планировщика;
- перехват системных вызовов;

- анализ списка таблиц хэндлов;
- сканирование PspCidTable;
- перехват SwapContext.

Скрытые процессы выявляются путем сравнения данных, полученных при помощи вышеописанных вариантов, с возвращаемыми функцией CreateToolhelp32Snapshot, выбранными в качестве эталонных.

2.4. Сигнатурное сканирование

Перед запуском процесса система производит сканирование исполняемого файла на предмет наличия известных вирусных сигнатур. В случае обнаружения совпадений запуск процесса блокируется. Стоит отметить, что разработка сигнатур до сих пор является процессом, тяжело поддающимся автоматизации [4]. Виной тому нарастающий полиморфизм и метаморфизм вирусов и червей, что делает синтаксические сигнатуры бессмысленными. Таким образом, для поддержания эффективности сигнатурных сканеров разработчики вынуждены помещать в базу десятки сигнатурных кодов, ассоциированных с одним и тем же вирусом. Очевидно, что проблема автоматизации выделения конкурентоспособных сигнатур достойна отдельного исследования и выходит за рамки поставленных в данной работе задач. Однако полностью отказаться от использования сигнатурного метода обнаружения вредоносного кода невозможно, ввиду его высокой эффективности, применительно к давно известным угрозам. Для повышения общей защищенности системы были использованы сигнатурные базы открытого антивируса CalmAV.

2.5. Целостность PE образа

Для выявления замаскированного вредоносного кода, перед запуском приложения, система производит проверку структуры исполняемого файла. Проверка состоит из следующих шагов:

- анализ таблицы импорта;
- анализ секции данных (проверка на наличие кода);
- проверка избыточности циклической суммы (CRC32);
- эмуляция загрузки файла с последующей проверкой соответствия структуры образа в памяти.

2.6. Работа с файловой системой и сетью

Для регистрации всех обращений к файловой системе в драйвер мониторинга был включен функционал драйвера фильтра файловой системы, основной задачей которого является перехват IRP-пакетов с командами IRP_MJ_CREATE.

Тот факт, что драйвер фильтра занимает в иерархии более высокий уровень, нежели драйвер файловой системы, позволяет ему модифицировать поток между приложениями и драйвером файловой системы.

Для регистрации сетевых соединений используется перехват события IRP _MJ _DEVICE_CONTROL для системных устройств \\DEVICE\\RAWIP, \\DEVICE \\UDP, \\DEVICE \\TCP, \\DEVICE\\IP.

Как уже было сказано выше, обработка данных, получаемых драйвером, осуществляется в отдельных модулях, работающих в пользовательском режиме.

3. Полученные результаты

Тестирование работоспособности системы проведено на серии известных вредоносных программ. Для сравнения использовались две идентично сконфигурированные виртуальные машины: VirtualBox, Windows XP SP2 с включенным стандартным сетевым экраном. Обеим машинам был обеспечен доступ в сеть Internet.

В процессе тестирования одна из машин все время оставалась «здоровой», другая «заражалась» через некоторое время после начала эксперимента. Ниже приведены сравнительные графики сетевой активности, а также интенсивности обращения к жесткому диску, для «здоровой» и «зараженной» машин.

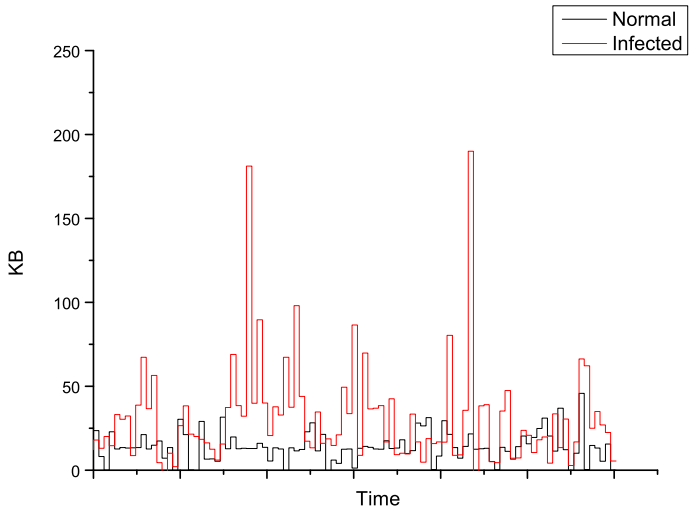


Рис. 2. Сетевая активность

В качестве известной вредоносной программы был использован троянский конь Trojan-Downloader.Win32.Agent.bet (здесь и далее названия приведены в соответствии с терминологией лаборатории Касперского). Как видно из графиков (рис. 2 и рис. 3), «зараженная» машина отличается повышенной сетевой активностью, а также увеличением загрузки центрального процессора. Кроме того, были зафиксированы попытки сокрытия процесса, попытки записи в си-

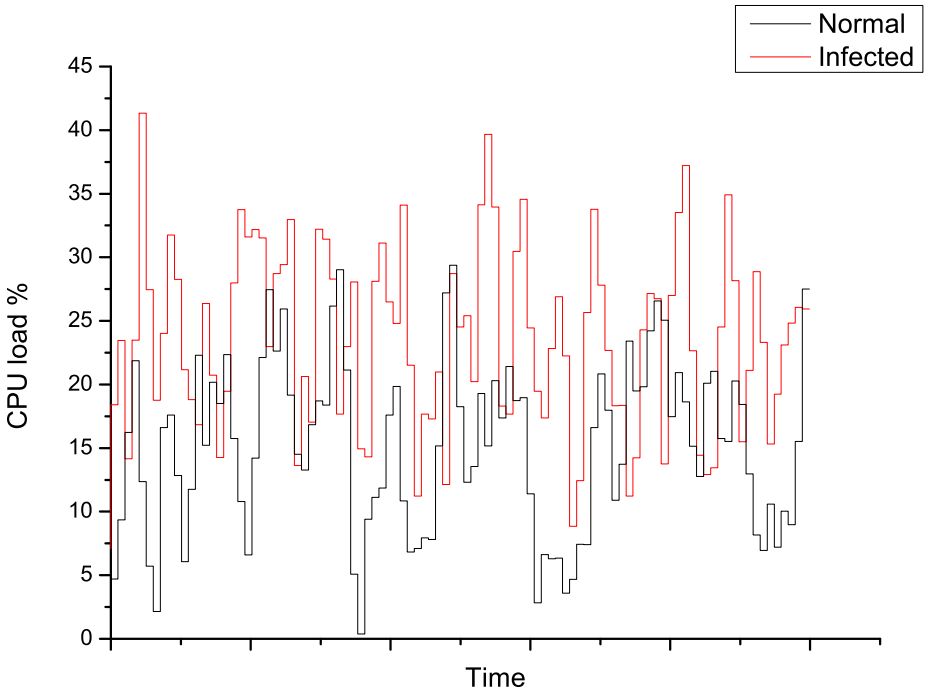


Рис. 3. Загрузка центрального процессора

стемные каталоги и реестр процессом, не производившим подобные действия во время обучения.

3.1. Ошибки второго рода

Большинство случаев несрабатывания системы относится к ситуациям, когда заражение приложения происходит в результате переполнения буфера (речь идет о приложениях и процессах, имеющих доступ к сети). В качестве примера можно привести поведение Trojan-Ransom.Win32.Hexzone.aer, прдставляющего собой Internet Explorer Browser Helper Object (BHO). Ситуация усугубляется наличием у некоторых процессов привилегий, необходимых для работы с реестром, что позволяет вредоносному коду закрепиться в системе.

К трудно обнаружимым вторжениями также относятся те, в которых проявления вредоносной программы минимальны (например бэкдоры, способные сохранять минимальную сетевую активность до получения команд извне). Ниже приводится сравнительный график сетевой активности для «здоровой» машины, а также машины, зараженной троянским конем Backdoor.Win32.VB.bdg (рис.4).

Однако серия успешно проведенных экспериментов позволяет говорить о высокой эффективности используемой модели. На данный момент число успешных индикаций зараженного состояния составляет не менее 73%.

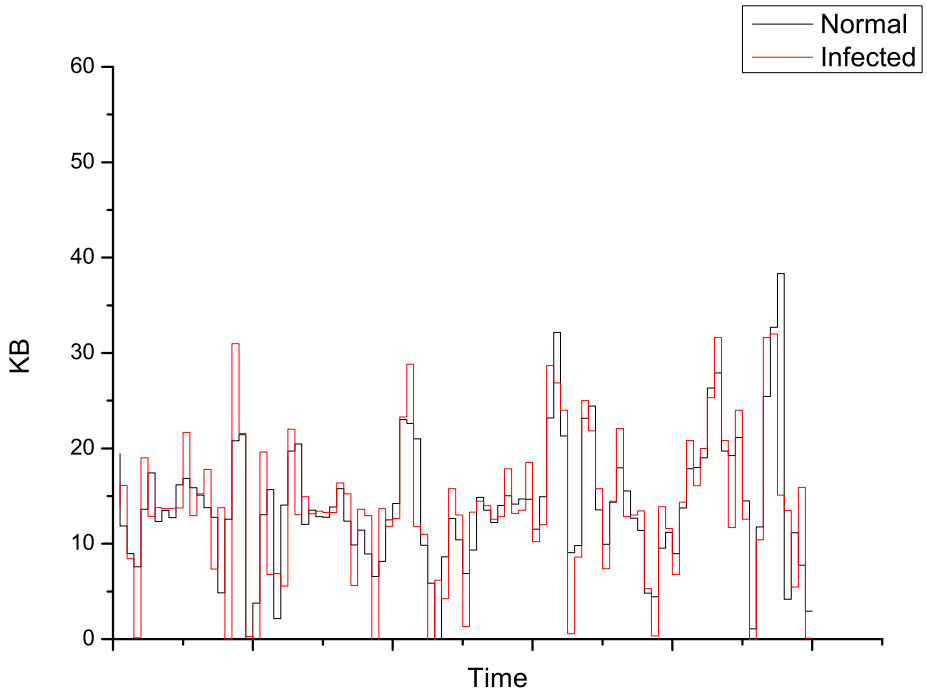


Рис. 4. Сетевая активность

Литература

1. Tarakanov A. O., Skormin V. A., Sokolova S. P. Immunocomputing: Principles and Applications. NY: Springer, 2003.
2. Иммунология / Хаитов Р.М. [и др.]. М.: Медицина, 2009. 320 с.
3. Дасгупта В. Искусственные иммунные системы. М.: Физматлит, 2006. 344 с.
4. Newsome J., Karp B., Song D. Polygraph: Automatically Generating Signatures for Polymorphic Worms. URL: http://www.ece.cmu.edu/ dawnsong/papers/polygraph.pdf (дата обращения: 01.02.2010).
5. Платонов В.В. Программно-аппаратные средства обеспечения информационной безопасности вычислительных сетей. М.: Академия, 2007. 240 с.
6. Руссинович М., Соломон Д. Внутреннее устройство Microsoft Windows: Windows Server 2003, Windows XP, Windows 2000. Санкт-петербург: Питер, Русская редакция, 2008. 992 с.
7. Peakman M., Vergani D. Basic and clinical immunology. NY: Churchill Livingstone, 1997.
8. Shibli A., Mwakalinga J., Muftic S. MagicNET: The Human Immune System and Network Security System // IJCSNS. 2009. V. 9, N. 1. P. 87–94.
9. Automatic Virus Analysis in the Digital Immune System. IBM T. J. Watson Research Center, 2000. URL: http://www.scribd.com/doc/24058575/Automatic-Virus-Analysis-in-the-Digital-Immune-System (дата обращения: 10.08.2009).

Математические структуры и моделирование
2010, вып. 21, с. 113–118

УДК 621.394

МОДЕЛИРОВАНИЕ СИСТЕМЫ ПРОТИВОДЕЙСТВИЯ DOS-АТАКАМ

С.В. Белим, С.Ю. Белим

В работе проводится моделирование системы противодействия DOS-атакам с помощью случайного уничтожения пакетов во входном буфере. Исследуется несколько возможных режимов работы системы.

Введение

Одним из трех аспектов информационной безопасности является доступность, то есть отклик системы на запрос за заданный промежуток времени. Для обеспечения доступности компьютерной системы необходимо принимать меры по обеспечению устойчивости системы к всплеску интенсивности поступающей информации. Как правило, соответствующие службы учитываются уже при проектировании системы. Принято говорить об архитектуре системы обработки входящих пакетов.

Простейшей является архитектура OQ (Output Queuing), в которой используется несколько входных линий и столько же обработчиков пакетов, причем каждый обработчик работает со своей входной линией. Архитектура OQ не подразумевает никаких дополнительных средств обеспечения доступности, кроме расчета устойчивого режима с помощью теории массового обслуживания.

В качестве примера можно привести архитектуру VOQ (Virtual Output Queuing), которая предполагает наличие алгоритма посылки поступающих пакетов обработчикам по циклу [1]. Другая архитектура, рассматриваемая в работах [2, 3], основывается на том, что 90% трафика составляет протокол TCP и сосредотачивается на контроле TCP-пакетов.

Данная статья посвящена моделированию состояния буфера входящих пакетов в рамках архитектуры FOQ (Feedback Output Queuing), предложенной в работе [4]. Данная архитектура предполагает обратную связь обработчика пакетов с входным буфером. Обработчики следят за состоянием буфера и уничтожают пакеты для предотвращения переполнения.

Омский государственный университет им. Ф.М. Достоевского.
E-mail: sbelim@omsu.ru

1. Постановка задачи

Рассмотрим компьютерную систему, обрабатывающую пакеты, поступающие во входной буфер длины L. Рассмотрение начнем с простой ситуации поступления пакетов с постоянной скоростью u. Более точно, пусть в буфер в единицу времени поступает u пакетов. Как обычно при моделировании компьютерных систем, время считаем дискретным, отсчитываемым по системному таймеру. Интервал между двумя «тиками» таймера примем за единицу. Будем считать, что поступающие пакеты обрабатываются системой с постоянной скоростью w пакетов в единицу времени. Обозначим количество пакетов в буфере в момент времени n через x_n, тогда без дополнительных систем уничтожения пакетов состояние буфера описывается разностным уравнением

$$x_{n+1} - x_n = u - w.$$

Или, вводя обозначение $v = u - w$, получаем рекуррентное соотношение

$$x_{n+1} = x_n + v.$$

Таким образом, задача разбивается на два случая: $v <= 0$ и $v > 0$. В первом случае ($v <= 0$) дополнительные средства очистки буфера не нужны, так как $x_{n+1} >= x_n$ для любого момента времени n. Этот результат достаточно очевиден, так как в этом случае скорость обработки пакетов не меньше скорости поступления пакетов ($u <= w$), и, как следствие, не происходит переполнение буфера. Обратный случай ($v > 0$), наоборот, приводит к росту количества пакетов в буфере с течением времени, что приводит к переполнению буфера. Как известно, такой способ нарушения работоспособности компьютерной системы получил название DOS-атаки.

Один из возможных способов борьбы с переполнением буфера был предложен в работе [7]. Основная идея метода состоит в уничтожении случайно выбранных пакетов во входном буфере в случае возрастания скорости поступления пакетов. Безусловно, при таком подходе существует не нулевая вероятность уничтожения «полезных» пакетов. Однако при отсутствии ответа на запрос серверы посылают повторный запрос. Вероятность же случайного уничтожения всех пакетов одного сервера достаточно мала.

2. Постоянный поток пакетов

Для рассматриваемой модели со случайным уничтожением пакетов рекуррентное соотношение примет вид:

$$x_{n+1} = x_n + v - d_n.$$

Здесь $v > 0$, а d_n – количество случайно уничтожаемых пакетов в момент времени n. Случайно уничтожаемые пакеты отбрасываются без обработки.

Существенным вопросом является выбор последовательности d_n, которая, очевидно, должна зависеть от заполнения буфера. Рассмотрим несколько возможных случаев:

1. $d_n = \alpha x_n$ – количество уничтожаемых пакетов прямо пропорционально количеству пакетов в буфере ($0 < \alpha < 1$). Рекуррентное соотношение будет иметь вид:

$$x_{n+1} = (1 - \alpha)x_n + v.$$

Для его решения необходимо составить соответствующее однородное уравнение. Выпишем состояние буфера в предыдущий момент времени:

$$x_n = (1 - \alpha)x_{n-1} + v$$

и, выразив из него v, подставим в предыдущее рекуррентное соотношение:

$$x_{n+1} = (2 - \alpha)x_n - (1 - \alpha)x_{n-1}.$$

Общее решение данного уравнения имеет вид:

$$x_n = C_1 + C_2(1 - \alpha)^n,$$

где C_1 и C_2 – константы. Начальное состояние системы – $x_0 = 0$. Для момента времени $n = 1$ из рекуррентного соотношения получаем значение $x_1 = v$. Можем найти частное решение, удовлетворяющее начальным условиям:

$$x_n = \frac{v}{\alpha}(1 - (1 - \alpha)^n).$$

В пределе больших n получаем

$$\lim_{n->\infty} x_n = \frac{v}{\alpha}.$$

Причем для всех n переполнение буфера не будет происходить, если для всех моментов времени n выполняется $x_n < L$, откуда получаем условие на коэффициент α:

$$\alpha > \frac{v}{L}.$$

2. $d_n = \alpha(x_n - x_{n-1})$ – количество уничтожаемых пакетов прямо пропорционально количеству поступающих в буфер пакетов ($0 < \alpha < 1$). Рекуррентное соотношение будет иметь вид:

$$x_{n+1} = (1 - \alpha)x_n + v - \alpha x_{n-1}.$$

Для его решения необходимо составить соответствующее однородное уравнение. Выпишем состояние буфера в предыдущий момент времени:

$$x_n = (1 - \alpha)x_{n-1} + v - \alpha x_{n-2}$$

и, выразив из него v, подставим в предыдущее рекуррентное соотношение:

$$x_{n+1} = (2 - \alpha)x_n - (1 - 2\alpha)x_{n-1} - \alpha x_{n-2}.$$

Общее решение данного уравнения имеет вид:

$$x_n = C_1 + C_2 n + C_3(-\alpha)^n,$$

где C_1, C_2 и C_3 – константы. Начальное состояние системы – $x_0 = 0$. Для момента времени $n = 1$ $x_1 = u$, для $n = 2$ из рекуррентного соотношения получаем значение $x_2 = u + v$. Можем найти частное решение, удовлетворяющее начальным условиям:

$$x_n = \frac{u - v}{(1+\alpha)^2} + \frac{2\alpha u + v}{(1+\alpha)} n + \frac{v - u}{(1+\alpha)^2}(-\alpha)^n.$$

Несложно заметить, что x_n растет со временем, однако не монотонно за счет слагаемого $(-\alpha)^n$. Тем не менее для любого заданного размера буфера L существует момент времени k такой, что $x_k > L$.

3. Случайный поток пакетов

Рассмотрим случай, в котором количество прибывающих пакетов v_n, а следовательно, и количество пакетов в буфере x_n являются случайной величиной с гауссовым распределением. Выпишем средние значения случайной величины:

$$< v_n >= v, \ < v_n v_k >= \delta_{nk}\sigma_v,$$

где δ_{nk} – символ Кронекера.

Найдем соответствующие средние значения для количества пакетов в буфере. Выберем количество уничтожаемых пакетов в виде $d_n = \alpha x_n$. Тогда рекуррентное соотношение примет вид:

$$x_n = (1 - \alpha)x_{n-1} + v_n.$$

Усреднение его по времени приводит к выражению

$$< x_n >= (1 - \alpha) < x_{n-1} > + v.$$

Отсюда можно получить выражение

$$< x_n >= \frac{v}{\alpha}(1 - (1 - \alpha)^n).$$

То есть при надлежащем выборе коэффициента α, как и в случае постоянного потока пакетов, можно добиться стабильной работы системы без переполнения буфера. Однако при случайной интенсивности поступления пакетов возможны резкие всплески количества пакетов в буфере, поэтому необходимо также оценить дисперсию случайной величины x_n:

$$(x_{n+1})^2 = (1 - \alpha)^2 (x_n)^2 + 2(1 - \alpha)x_n v_n + (v_n)^2.$$

Выражая v_n из рекуррентного выражения для x_n, получаем следующее соотношение:

$$(x_{n+1})^2 = (1 - \alpha)^2 (x_n)^2 + 2(1 - \alpha)(x_n)^2 - 2(1 - \alpha)^2 x_n x_{n-1} + (v_n)^2.$$

Усредняя это выражение по времени, получаем

$$< (x_{n+1})^2 >=$$

$$= (1-\alpha)^2 < (x_n)^2 > +2(1-\alpha) < (x_n)^2 > -2(1-\alpha)^2 < x_n x_{n-1} > + < (v_n)^2 > .$$

Предполагая, что x_n имеет гауссово распределение со средними значениями:

$$< x_n >= \frac{v}{\alpha}(1-(1-\alpha)^n, \ < x_n x_k >= \sigma_x \delta_{nk}.$$

Откуда получаем

$$\sigma_x = \frac{\sigma_v}{2-(2-\alpha)^2}.$$

Следовательно, дисперсия наполнения буфера прямо пропорциональна дисперсии интенсивности поступающих пакетов.

По хорошо известному из теории вероятностей правилу «трех сигм» для нормального распределения с вероятностью 0.9973 случайная величина x_n будет попадать в интервал $[< x_n > -3\sigma_x, < x_n > +3\sigma_x)$. Следовательно, чтобы с вероятностью 0.9973 не происходило переполнение буфера, необходимо, чтобы $< x_n > +3\sigma_x < L$. Отсюда получаем более жесткое, чем в случае постоянного потока, условие на коэффициент α:

$$\frac{v}{\alpha} + 3\frac{\sigma_v}{2-(2-\alpha)^2} < L.$$

Данное неравенство сводится к неравенству третьей степени, которое всегда имеет решение. Значение параметров входящего потока v и σ_v может быть определено экспериментально. Соответственно настройка системы производится выбором значения α. Рассмотренные соотношения не позволяют получить точное оптимальное значение параметра α, а только определяют его граничные значения.

4. Заключение

Таким образом, задача построения системы защиты от DOS-атак с помощью механизма уничтожения случайным образом входящих пакетов во входном буфере разрешима. Степень надежности системы может варьироваться с помощью изменения параметра системы, отвечающего за активность уничтожения пакетов. При этом следует учитывать, что активность системы защиты снижает скорость обработки информации. Но это общеизвестный факт, касающийся всех систем защиты, к нему уже все привыкли и он не считается недостатком. Кроме того система случайного уничтожения входящих пакетов не является гарантированной защитой от DOS-атак. Но и в этом случае мы сталкиваемся с общеизвестным фактом: гарантированной защиты не бывает в принципе.

Литература

1. Nong G., Hamdi M. On the provisioning of Quality of Service guarantees for input queud switches //IEEE Communications Magazine. 2000. V. 38(12). P.62–69.
2. Jacobson V. Congestion avoidance and control //Proceeding of ACM SIGCOMM'88. 1988. Stanford. P.314–329.
3. Stevens W. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms // IETF RFC 2001, January 1997.
4. Firoiu V., Zhang X., Gunduzhan E., Christin N. Providing service guarantees in high-speed switching systems with feedback output queuing. // arXiv:cs/0406019v1.

Математические структуры и моделирование
2010, вып. 21, с. 119–127

УДК 65.012.810

ОБРАТНАЯ ЗАДАЧА ПОСТРОЕНИЯ МАНДАТНОЙ ПОЛИТИКИ БЕЗОПАСНОСТИ

С.В. Белим, Н.Ф. Богаченко, И.А. Фирдман

В данной работе проводится исследование возможности построения мандатной политики безопасности для компьютерных систем на основе известных правил разграничения доступа.

Введение

Разработка политики безопасности является одной из первых задач, которые приходится решать при построении защищенной системы обработки информации. При этом политика безопасности может строиться как неформально, в виде правил и инструкций, так и формально, в виде строгой математической модели. Несмотря на то, что неформальный подход проще для разработки и внедрения, он существенно уступает формальному в надежности, так как не допускает строгих доказательств гарантированной защищенности. Данное обстоятельство нашло отражение в различных документах, определяющих классы защищенности компьютерных систем. Так, например, в «Оранжевой книге» системы, имеющие неформальную политику безопасности, относятся к классам группы «С», а имеющие формальную политику безопасности — к высшему классу «А1» [8].

Выбор политики безопасности осуществляется исходя из задач, стоящих перед системой защиты. Выработка правил безопасности должна начинаться с четкого определения понятия «угроза безопасности» в контексте заданной системы. С другой стороны, знание злоумышленником политики безопасности системы позволяет выявить слабые места подсистемы защиты. Поэтому политика безопасности должна оставаться максимально закрытой информацией. Однако полностью скрыть правила безопасности невозможно в силу наличия утечки информации через внутренних сотрудников. Также злоумышленник может выявлять элементы политики безопасности экспериментальным путем, выделяя разрешенные и запрещенные потоки.

На сегодняшний день широкое распространение получили три вида формальных политик безопасности – дискреционная, мандатная и ролевая [3, 5]. В данной статье мы ограничимся рассмотрением мандатной политики безопасности и возможностями ее анализа злоумышленником. Основой мандатной поли-

Омский государственный университет им. Ф.М. Достоевского.
E-mail: sbelim@mail.ru

тики безопасности является решетка ценностей. Напомним, что *решетка* – это частично упорядоченное множество, в котором каждое двухэлементное подмножество имеет как наименьшую верхнюю (sup), так и наибольшую нижнюю (inf) грани, принадлежащие этому множеству [4, с. 17].

1. Постановка задачи

Как уже было сказано во введении, мандатная политика безопасности строится на базе решетки ценностей, то есть обычной алгебраической решетки, элементы которой играют роль меток безопасности объектов и субъектов компьютерной системы [5]. Обычно задача построения мандатной политики безопасности формулируется следующим образом: задана решетка ценностей L и отображение C множества субъектов $\mathbf{S}$ и объектов $\mathbf{O}$ на решетку

$$C : \mathbf{S} \times \mathbf{O} \to L.$$

Окончательно политика безопасности формулируется определением правила P – сопоставления меток безопасности при обращении на доступ субъекта к объекту. После чего исследуются разрешенные и запрещенные каналы передачи информации. В дальнейшем такой подход будем называть ***прямой задачей*** построения мандатной политики безопасности.

Однако в практике защиты информации часто приходится решать задачу в другой постановке: для системы определены разрешенные и (или) запрещенные каналы передачи информации, требуется построить соответствующую мандатную политику безопасности, то есть решетку ценностей L, отображение C и правило P, приводящую к такому же разделению потоков информации. Такую постановку проблемы будем называть ***обратной задачей*** построения мандатной политики безопасности.

Необходимость решения обратной задачи построения мандатной политики безопасности возникает, как правило, в двух случаях. Во-первых, при формальном построении правил разграничения доступа для организации с уже сложившимися потоками информации, причем нарушение или изменение потоков существенно влияет на работоспособность организации. Во-вторых, при исследовании некоторой системы на наличие каналов утечки информации методом «черного ящика», когда с помощью проб и ошибок можно выявить разрешенные или запрещенные потоки информации и, на их основе, строить предположения о применяемой политике безопасности.

Очевидно, что если две политики безопасности определяются одинаковыми параметрами (L, C, P), то они приводят к одинаковым множествам запрещенных и разрешенных потоков, в противном случае политика безопасности приводила бы к неоднозначным результатам функционирования, что недопустимо в практике защиты информации. Верно и обратное утверждение. Если две политики безопасности определяются двумя одинаковыми параметрами (L, C), или (L, P), или (C, P) и одинаковыми множествами запрещенных и разрешенных потоков, то и третий параметр политики безопасности будет одинаков. Верность этого утверждения также следует из однозначности политики безопасности.

Рассмотрим один из возможных путей решения обратной задачи построения мандатной политики безопасности в рамках субъектно-объектного подхода [5], сводящийся к исследованию различных алгебраических решеток и соответствующих им ориентированных графов. Мандатная политика безопасности определяется тремя параметрами. Будем варьировать два из них произвольным образом и определять третий. Согласно вышесказанному, такое решение будет единственным.

В качестве подбираемого параметра выберем решетку ценностей L. Сделаем предположения относительно отображения C. Пусть в системе каждому субъекту и каждому объекту сопоставляется ровно один элемент из решетки L, однако одному элементу из L может сопоставляться сколько угодно субъектов и объектов компьютерной системы. В качестве правил безопасности выберем разрешение потоков только снизу вверх: доступ $S \xrightarrow{p} O$ разрешен, если:

1) $p = read$ и $C(S) \geqslant C(O)$,
2) $p = write$ и $C(S) \leqslant C(O)$.

Для простоты изложения будем считать, что субъекты компьютерной системы также являются и объектами компьютерной системы. Применим следующий алгоритм построения ориентированного графа G:

1. Каждому объекту сопоставим вершину графа.
2. Если между двумя объектами системы возможны потоки информации в обе стороны, то соответствующие две вершины объединим в одну вершину.
3. Если между двумя объектами системы возможен поток информации только в одну сторону, то добавим в графе соответствующую ориентированную дугу.

Построенный граф задает отношение частичного нестрогого порядка на множестве объектов. Очевидно, что полученное множество не всегда будет являться алгебраической решеткой. В связи с чем можно сформулировать две разновидности задач построения решетки ценностей.

1. ***Полная задача***. Граф G известен полностью и необходимо определить соответствующую ему решетку. В этом случае предполагается, что в системе реализована мандатная политика безопасности и по разрешенным потокам информации решетка ценностей может быть полностью восстановлена.
2. ***Неполная задача***. Граф G известен не полностью, необходимо определить минимальную решетку, включающую его. Предполагается, что в системе реализована мандатная политика безопасности, однако по некоторым причинам известны не все разрешенные и запрещенные потоки.

В данной статье мы ограничимся решением только полной обратной задачи построения мандатной политики безопасности. Полная обратная задача построения мандатной политики безопасности сводится к построению решетки, диаграмма которой изоморфна графу G. В первую очередь необходимо проверить, является ли граф G решеточным [2].

Определение 1. *Решеточным графом* будем называть ориентированный граф, вершины которого образуют решетку, при этом:

- отношение порядка задается ориентированными путями: если в графе существует ориентированный путь $p(r_1, r_2)$, то $r_1 \geqslant r_2$;

- $r = \sup(r_1, r_2) \Longleftrightarrow$

  1. $\exists\ p(r, r_1)\ \&\ p(r, r_2)$, то есть r является верхней гранью.
  2. Если $\exists\ p(r', r_1)\ \&\ p(r', r_2)$, то $\exists\ p(r', r)$, то есть r минимальна среди всех верхних граней.

- $r = \inf(r_1, r_2) \Longleftrightarrow$

  1. $\exists\ p(r_1, r)\ \&\ p(r_2, r)$, то есть r является нижней гранью.
  2. Если $\exists\ p(r_1, r')\ \&\ p(r_2, r')$, то $\exists\ p(r, r')$, то есть r максимальна среди всех нижних граней.

Задача проверки решеточности графа очевидно может быть решена прямым перебором.

Теорема 1. *Трудоемкость проверки решеточности графа прямым перебором не превышает* $O(n^4)$.

Доказательство. Пусть ориентированный граф задан матрицей смежности M размерности $n \times n$. Как будет показано ниже, для проверки решеточности графа следует вычислить *матрицу достижимости* M^*. Согласно [7, с. 176], элемент m^*_{ij} матрицы достижимости равен 1, если в орграфе существует ориентированный путь из вершины r_i в вершину r_j, и 0 – в противном случае. Трудоемкость *алгоритма Уоршелла*, вычисляющего матрицу достижимости ориентированного графа, равна $O(n^3)$ [6, с. 48].

Оценим трудоемкость поиска sup для заданной пары вершин r_i и r_j. Обозначим эту величину $T(\sup)$. Имея в распоряжении матрицу достижимости, за $O(n)$ шагов выбираются вершины, связанные одновременно ориентированными путями с вершинами r_i и r_j (это те вершины r_k, для которых $m^*_{ki} \cdot m^*_{kj} = 1$). Среди отобранных вершин выполняется проверка того, что существует одна и только одна вершина, в которую можно попасть из оставшихся. В худшем случае это можно осуществить за $O(n^2)$ шагов, используя все ту же матрицу достижимости. Тогда трудоемкость $T(\sup) = O(n) + O(n^2) = O(n^2)$.

Очевидно, что поиск inf имеет ту же трудоемкость, что и поиск sup, если в графе инвертировать ориентацию дуг. Для расчетов достаточно транспонировать матрицу достижимости.

Проверка того, что орграф является решеточным, сводится к проверке существования inf и sup для каждой пары вершин. Трудоемкость перебора всех пар равна $O(n^2)$.

В итоге искомая трудоемкость равна $O(n^3) + O(n^4) = O(n^4)$. ■

2. Некоторые решения полной задачи

На сегодняшний день в моделях безопасности компьютерных систем получили широкое распространение три решетки: линейная решетка, решетка подмножеств и MLS-решетка, представляющая собой декартово произведение первых двух [5, с. 14]. Рассмотрим подробно первые две решетки и алгоритмы проверки изоморфности им заданного решеточного графа.

При построении алгоритмов проверки изоморфности необходимо учитывать тот факт, что одной и той же решетке может соответствовать несколько решеточных графов. Как показано в [2], для произвольной решетки существует изоморфный ей решеточный граф, но он не единственен.

Определение 2. Решеточные графы, изоморфные одной и той же решетке, назовем *эквивалентными*.

Определение 3. *Эквивалентные преобразования* решеточного графа – это такие операции над графом, которые оставляют граф решеточным и не меняют изоморфную ему решетку.

Определение 4. Назовем решеточный граф *транзитивным*, если в нем дуга (r_1, r_2) существует тогда и только тогда, когда $r_1 \geqslant r_2$.

Заметим, что решеточный граф, в котором дуга (r_1, r_2) существует, тогда и только тогда, когда не существует ориентированного пути $p(r_1, r_2)$, такого, что $|p(r_1, r_2)| > 1$ (состоящего более чем из одной дуги) называется *диаграммой Хассе* [7, с. 80].

Определение 5. Дуга (r_1, r_2) решеточного графа называется *транзитивной*, если существует вершина r, отличная от r_1 и r_2, такая, что $r_1 \geqslant r$ (существует ориентированный путь $p(r_1, r)$) и $r \geqslant r_2$ (существует ориентированный путь $p(r, r_2)$).

С учетом данных определений можно сформулировать следующие очевидные утверждения.

Предложение 1. *В диаграмме Хассе отсутствуют транзитивные дуги.*

Предложение 2. *Добавление или удаление транзитивной дуги является эквивалентным преобразованием решеточного графа.*

Предложение 3. *Эквивалентный транзитивный решеточный граф можно получить из диаграммы Хассе, пополнив ее всевозможными транзитивными дугами.*

Предложение 4. *Матрица достижимости транзитивного решеточного графа совпадает с его матрицей смежности.*

Предложение 5. *Пусть решеточный граф задан матрицей смежности. Тогда алгоритм Уоршелла строит матрицу смежности транзитивного решеточного графа, эквивалентного исходному.*

Доказательство. Алгоритм Уоршелла строит матрицу достижимости. Интерпретация этой матрицы как матрицы смежности заключается в добавлении к графу всех транзитивных дуг, то есть в построении транзитивного решеточного графа, эквивалентного исходному. ■

Далее будем считать, что ориентированный решеточный граф на n вершинах задан матрицей смежности M.

2.1. Линейная решетка

Под линейно упорядоченным множеством понимается множество (назовем его SL), для любых двух элементов которого определено отношение порядка. Как легко показать, такое множество образует решетку. Наименьшая верхняя и наибольшая нижняя грани для любых двух элементов определяются достаточно просто. Возьмем два элемента $a, b \in SL$, без потери общности будем считать, что $a \geqslant b$, тогда $\sup(a,b) = a$ и $\inf(a,b) = b$. Решетка данного вида является наиболее распространенной в системах защиты информации, она описывает уровни доступа к данным.

Теорема 2. *Трудоемкость проверки изоморфности решеточного графа некоторой линейной решетке не превосходит* $O(n^3)$.

Доказательство. Для проверки изоморфности решеточного графа линейной решетке достаточно показать, что любая пара вершин сравнима, то есть для любой пары вершин существует ориентированный путь, их соединяющий. Как и в теореме 1, можно воспользоваться матрицей достижимости M^*: r_1 и r_2 сравнимы тогда и только тогда, когда либо $(m_{ij} = 1)$, либо $(m_{ji} = 1)$ (одновременно равенство единице выполняться не может, так как решеточный граф не содержит ориентированных циклов [2]). Трудоемкость такой проверки $O(n^2)$. Но трудоемкость вычисления матрицы M^* равна $O(n^3)$. ■

Следствие 2.1. *Трудоемкость проверки изоморфности транзитивного решеточного графа некоторой линейной решетке равна* $O(n^2)$.

Доказательство. Пусть решеточный граф является транзитивным. Тогда согласно предложению 4 его матрица смежности совпадает с матрицей достижимости. Отсюда следует, что трудоемкость проверки изоморфности графа линейной решетке равна $O(n^2)$ (так как достаточно для каждой пары вершин выполнить проверку: $(m_{ij} = 1)$ или $(m_{ji} = 1)$). ■

2.2. Решетка подмножеств

Пусть задано множество X. Рассмотрим множество всех его подмножеств $SX = \{A | A \subseteq X\}$. Введем на множестве SX отношение порядка: $\forall A, B \in SX : \ A \geqslant B \Leftrightarrow A \supseteq B$. Легко доказать, что отношение $A \supseteq B$ есть отношение нестрогого порядка, а множество SX частично упорядоченно: так не для любой пары элементов определено отношение порядка. Наибольшую

нижнюю и наименьшую верхнюю грани определим на основе операций пересечения и объединения множеств: $\sup(A, B) = A \cup B$, $\inf(A, B) = A \cap B$. Также нетрудно доказать, что $(SX, \supseteq, \cup, \cap)$ – алгебраическая решетка.

Примерами использования данной решетки в реальных компьютерных системах может служить множество атрибутов файла и зависимость выходной величины от подмножества множества входных элементов.

Теорема 3. *Трудоемкость проверки изоморфности решеточного графа некоторой решетке подмножеств не превосходит* $O(n^3)$.

Доказательство. Алгоритм проверки изоморфности решеточного графа решетке подмножеств состоит из следующих этапов.

1. Проверяется, является ли число вершин графа степенью двойки (так как число узлов решетки подмножеств совпадает с мощностью булеана $\mathcal{B}(X)$ некоторого исходного множества $X = \{x_1, \ldots, x_k\}$ и равна 2^k, где $k = |X|$). Трудоемкость этого этапа не зависит от n.
2. Строится матрица достижимости M^*. Трудоемкость этого этапа $T_2 = O(n^3)$.
3. Каждой вершине графа сопоставляется некоторое подмножество множества X. Мощность каждого из этих подмножеств не превосходит k.

   Для формирования подмножеств на начальном шаге всем вершинам r_i приписывается множество $R_i = \varnothing$ (трудоемкость $O(n)$).

   Далее ищется сток (вершина без исходящих дуг) r_s. Согласно [2], такая вершина существует и единственна. Стоком является вершина, для которой строка в матрице смежности M состоит из одних нулей. Трудоемкость поиска стока $O(n^2)$.

   Затем выбираются вершины, имеющие одну и только одну исходящую дугу, причем эта дуга должна вести в сток (назовем их одноэлементными вершинами): для каждой такой вершины r_j в матрице смежности элемент $m_{js} = 1$, а остальные элементы этой строки – нулевые. Число одноэлементных вершин должно равняться $k = \log_2 n$. Трудоемкость поиска $O(n^2)$.

   Соответствующие одноэлементным вершинам множества пополняются: $R_{j_v} := R_{j_v} \cup \{x_v\}$ $(v = 1, \ldots, k)$. Трудоемкость пополнения $k \cdot T(\cup) = \log_2 n \cdot T(\cup)$, где $T(\cup)$ – трудоемкость операции объединения множеств.

   Окончательно множества, связанные с вершинами графа, формируются следующим образом: если элемент матрицы достижимости $m^*_{ij_v} = 1$ (существует путь из вершины r_i в одноэлементную вершину r_{j_v}), то $R_i := R_i \cup \{x_v\}$. Трудоемкость формирования множеств R_i для всех вершин равна $n \cdot k \cdot T(\cup) = n \cdot \log_2 n \cdot T(\cup)$.

   Будем считать, что множество реализовано при помощи двоичных векторов, тогда $T(\cup) = O(k) = O(\log_2 n)$ [1, с. 109]. Трудоемкость третьего этапа $T_3 = O(n) + O(n^2) + O(n^2) + O((\log_2 n)^2) + O(n \cdot (\log_2 n)^2) = O(n^2)$.
4. Проверяется, что $\{R_1, \ldots, R_n\} = \mathcal{B}(X)$. Очевидно, достаточно проверить, что все R_i $(i = 1, \ldots, n)$ различны. Трудоемкость этой процедуры $n^2 \cdot T(\lesseqgtr)$, где $T(\lesseqgtr)$ – трудоемкость операции сравнения множеств.

Если по-прежнему используется реализация множества при помощи двоичных векторов, то $T(\leqslant) = O(\log_2 n)$. Тогда трудоемкость четвертого этапа $T_4 = O(n^2 \cdot \log_2 n)$.

В итоге получаем, что результирующая трудоемкость представленного алгоритма проверки изоморфности решеточного графа решетке подмножеств равна $O(n^3) + O(n^2) + O(n^2 \cdot \log_2 n) = O(n^3)$. ■

Следствие 3.1. *Трудоемкость проверки изоморфности транзитивного решеточного графа некоторой решетке подмножеств равна* $O(n^2 \cdot \log_2 n)$.

Доказательство. Пусть решеточный граф является транзитивным. Тогда в алгоритме, представленном в доказательстве теоремы 3, этап 2 можно опустить, так как согласно предложению 4 в этом случае матрица смежности будет совпадать с матрицей достижимости. Отсюда следует, что трудоемкость проверки изоморфности графа решетке подмножеств равна $O(n^2 \cdot \log_2 n)$. ■

Таким образом, если есть дополнительная информация о типе используемой решетки, то задача восстановления политики безопасности упрощается.

Решетка подмножеств играет существенную роль при построении мандатной политики безопасности. Следующая теорема показывает, что любую мандатную политику безопасности можно свести к построенной на решетке подмножеств при условии, что будут использоваться не все элементы решетки. Такая ситуация возможна на практике, когда в качестве меток безопасности используется тематический классификатор, но при этом задействованы не все тематики.

Теорема 4. *Любая решетка изоморфна некоторой подрешетке решетки подмножеств.*

Доказательство. Пусть дана конечная решетка S с вершинами $\{a_0, a_1, \ldots, a_n\}$. Будем считать, что a_0 – нижняя грань всей решетки, a_n – верхняя грань всей решетки. В силу антисимметричности отношения порядка, в решетке S отсутствуют различные вершины a_i и a_j такие, что $a_i \leqslant a_j$ и $a_j \leqslant a_i$. Покажем, что в этом случае S изоморфна некоторой подрешетке решетки всех подмножеств множества $A = \{1, 2, \ldots, n\}$. Для этого сопоставим каждой вершине a_i множество $A_i = \{k \in A | a_k \leqslant a_i\}$ ($A_0 = \varnothing$, $A_n = A$). При этом очевидно $A_i \subseteq A_j$, если $a_i \leqslant a_j$. Обратно, если $A_i \subseteq A_j$, то из $i \in A_i$ следует $i \in A_j$ и $a_i \leqslant a_j$. Таким образом отображение $\varphi : S \to A$, определенное как $\varphi(a_i) = A_i$ $(i = 0, \ldots, n)$, сохраняет отношение порядка. Кроме того φ инъективно, поскольку из $A_i = A_j$ следует $A_i \subseteq A_j$ и $A_j \subseteq A_i$, затем $a_i \leqslant a_j$ и $a_j \leqslant a_i$, откуда следует $a_i = a_j$. Поэтому φ является вложением решетки S в решетку подмножеств множества A. ■

Таким образом, решение обратной задачи можно начинать с построения решетки подмножеств, а затем уже пытаться оптимизировать решение.

3. Заключение

Обратная задача построения мандатной политики безопасности разрешима в общем случае за полиномиальное время. Данный результат существенен при построении соответствующих программных комплексов, так как гарантирует получение результата за приемлемое время. Наличие дополнительной информации о типе используемой решетки существенно уменьшает время решения задачи.

Литература

1. Ахо А.В., Хопкрофт Д.Э., Ульман Д.Д. Структуры данных и алгоритмы. М.: Издательский дом «Вильямс», 2000. 384 с.
2. Белим С.В., Богаченко Н.Ф., Ракицкий Ю.С. Совместная реализация мандатного и ролевого разграничения доступа к информации в компьютерных системах // Математические структуры и моделирование. 2009. Омск: ООО «УниПак». Вып. 20. С. 141-152.
3. Гайдамакин Н.А. Разграничение доступа к информации в компьютерных системах. Екатеринбург: Изд-во Урал. ун-та, 2003. 328 с.
4. Гретцер Г. Общая теория решеток / Под ред. Д.М. Смирнова. М.: Мир, 1981. 456 с.
5. Девянин, П.Н. Модели безопасности компьютерных систем. М.: Издательский центр «Академия», 2005. 144 с.
6. Новиков Ф.А. Дискретная математика для программистов. СПб.: Питер, 2001. 304 с.
7. Хаггарти Р. Дискретная математика для программистов / Пер. с англ. М.: Техносфера, 2005. 400 с.
8. URL: http://www.securitylab.ru/informer/240650.php (дата обращения: 22.03.2010).

Математические структуры и моделирование
2010, вып. 21, с. 128–132

УДК 65.012.810

ОБЪЕДИНЕНИЕ МАНДАТНЫХ ПОЛИТИК БЕЗОПАСНОСТИ

С.В. Белим, Ю.С. Ракицкий

В данной работе рассмотрен возможный подход к построению мандатной политики безопасности составного предприятия. Исследована возможность непротиворечивого объединения уровней доступа. Предложен один из возможных алгоритмов.

1. Постановка задачи

Пусть организация состоит из N отделов D_1, $D_2, \ldots, D_n$. В каждом из отделов D_i $(i = \overline{1,N})$ действует мандатная политика безопасности, построенная на решетке L_i [2]. Необходимо построить общую политику безопасности организации, непротиворечиво включающую в себя все политики безопасности отделов. Данная задача распадается на два случая:

1. Политика безопасности изначально строится для всей организации, функционирование отделов должно быть организовано в определенной степени изолированно. Причем надо учитывать, что возможна ситуация, когда в каждом отделе свои требования к политике безопасности. Если организационными мерами возможно ввести для всех отделов единую шкалу ценности информации, то задача становится тривиальной.
2. Происходит объединение нескольких организаций в единую корпорацию, при этом начальные организации приобретают статус отделов. Необходимо учитывать, что в каждой организации до объединения существовала своя политика безопасности, изменение которой может привести к невозможности дальнейшего функционирования отдела. Таким образом, необходимо построить политику безопасности корпорации, сохранив при этом политики безопасности отделов.

В обоих случаях необходимо строить общую решетку ценностей, включающую в себя решетки отделов. Дополнительно необходимо обеспечить взаимодействие между отделами свободное от утечек информации. Будем решать задачу индуктивно. Рассмотрим объединение двух отделов. Далее, рассматривая полученное объединение как один новый отдел, присоединим к нему третий отдел и так далее. Таким образом, достаточно научиться строить объединение

Омский государственный университет им. Ф.М. Достоевского.
E-mail: sbelim@omsu.ru

двух отделов, которые без потери общности можно обозначить D_1 и D_2, а соответствующие им решетки ценностей L_1 и L_2. Документы, исходящие из одного отдела в другой, должны автоматически включаться в документооборот нового отдела. Следует отметить, что пересечение решеток может быть непустым $L_1 \cap L_2 \neq \emptyset$. Если передаваемый документ имеет метку безопасности из пересечения решеток, то проблем с переклассификацией не возникает, и, как следствие, отсутствует утечка информации. Поэтому в дальнейшем будем рассматривать только ситуацию, когда входящее сообщение имеет метку безопасности, отсутствующую в данном отделе.

2. Простое решение задачи

Наиболее простым решением задачи является построение единой решетки ценностей организации как декартова произведения решеток [3] отделов:

$$L = L_1 \times L_2.$$

При таком подходе каждый документ в системе будет характеризоваться парой меток безопасности (m_1, m_2) ($m_1 \in L_1$, $m_2 \in L_2$). В первом отделе при разграничении доступа учитывается только m_1, во втором – только m_2. Проблем с переклассификацией документов при передаче между отделами не возникает, так как происходит просто переключение с одной метки на другую.

Рассмотрим пример такого объединения. Пусть в отделе D_1 действует линейная решетка ценностей $L_1 = a_1, a_2, a_3$ с тремя уровнями безопасности $(a_1 < a_2 < a_3)$, диаграмма которой приведена на рисунке 1.

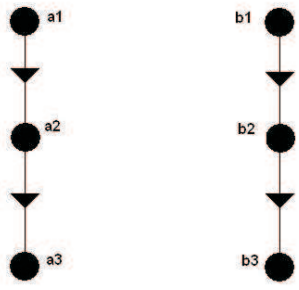


Рис. 1. Линейная решетка ценностей

Во втором отделе D_2 пусть действует линейная решетка ценностей $L_2 = b_1, b_2, b_3$ с тремя уровнями безопасности $(b_1 < b_2 < b_3)$, диаграмма которой также представлена на рисунке 1. Итоговая решетка безопасности объединенной организации будет иметь 9 меток безопасности, ее диаграмма приведена на рисунке 2.

Две метки безопасности в декартовом произведении сравнимы, если сравнимы как первая, так и вторая пара координат одновременно, причем они упорядочены одинаково.

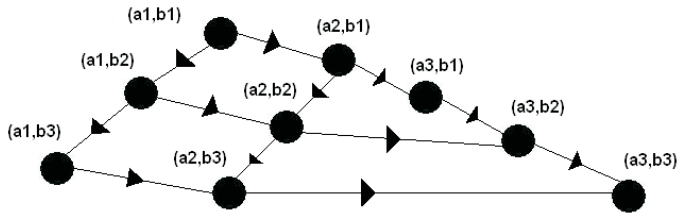


Рис. 2. Итоговая решетка безопасности объединенной организации

3. Возможности оптимизации решения

Проблема, возникающая при построении простого решения, заключается в обеспечении безопасности. Если метка безопасности сообщения из решетки L_1 отдела D_1 не входит в решетку D_2, то это означает, что ни один субъект отдела D_2 не может прочитать такое сообщение, пока ему не предоставят соответствующий доступ, то есть новую метку безопасности из декартова произведения решеток L_1 и L_2. При этом возникает ситуация, когда возможно отсутствие обмена между некоторыми субъектами из различных отделов, а это означает что ни одна из новых меток безопасности, полученных из декартова произведения решеток, им не подходит. Иначе будет возникать утечка информации, поскольку в полученной новой решетке самый низкий уровень a_3, b_3 предполагает доступ к информации каждого из отделов. Для того чтобы избежать подобной ситуации, необходимо дополнить новую решетку безопасности дополнительными уровнями безопасности, которые будут являться подрешетками новой решетки и имитировать работу каждого из отделов до слияния. Для этого необходимо ввести элементарное преобразование, которое будет дополнять любую решетку пустым элементом, или нулевым элементом. Возвращаясь к нашему примеру, решетка ценностей L_1 так и останется линейной, но минимальным элементом станет не a_3, а $\varnothing$. Аналогичные рассуждения можно применить и к решетке L_2. В результате получим решетки $L_1 \cup \varnothing = L_1^{\varnothing}$ и $L_2 \cup \varnothing = L_2^{\varnothing}$, диаграммы которых приведены на рисунке 3.

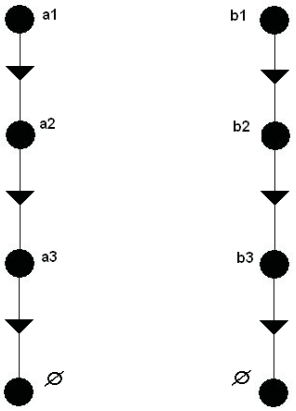


Рис. 3. Решетки $L_1 \cup \varnothing = L_1^{\varnothing}$ и $L_2 \cup \varnothing = L_2^{\varnothing}$

В случае если решетка ценностей не будет являться линейной, то, по определению решетки, для любых двух элементов существует наименьшая точная нижняя грань, то есть в любой решетке есть наименьший элемент. Это означает, что можно добавить пустой элемент, причем свойства решетки при этом не нарушатся. Пример такого дополнения изображен на рисунке 4.

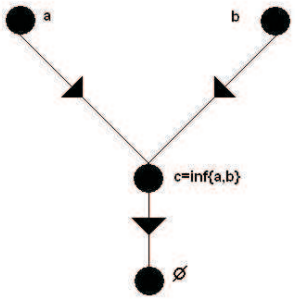


Рис. 4. Пример дополнения нелинейной решетки нулевым элементом

Построение единой решетки из двух полученных нами при помощи добавления пустого элемента можно также осуществить при помощи декартова произведения

$$L^{\varnothing} = L_1^{\varnothing} \times L_2^{\varnothing}.$$

При таком подходе результирующая решетка, диаграмма которой изображена на рисунке 5, будет состоять из 16-ти элементов. При этом можно заметить, что 9 элементов решетки $L^{\varnothing}$, образующие подрешетку, являются в точности решеткой L, диаграмма которой представлена на рисунке 1. Можно говорить о том, что решетка L является инструментом обеспечения информационного обмена между отделами D_1 и D_2, причем этот информационный обмен будет безопасным, поскольку для каждого вида взаимодействия предусмотрен специальный уровень безопасности. Кроме того, в решетке $L^{\varnothing}$ можно также выделить еще 2 подрешетки, каждая из которых будет имитировать работу отделов без информационного обмена. Диаграмма решетки $L^{\varnothing}$ представлена на рисунке 5.

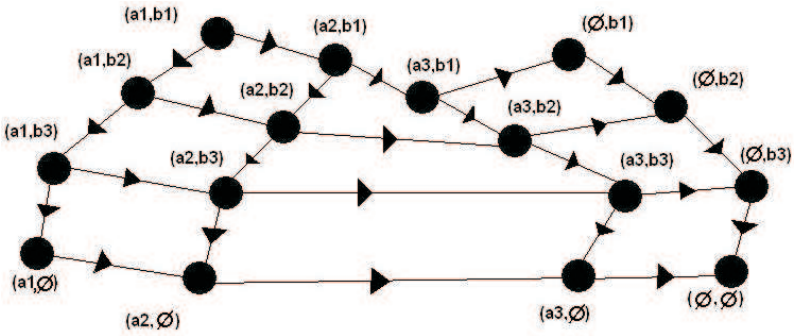


Рис. 5. Решетка $L^{\varnothing}$

4. Заключение

Таким образом, для построения общей политики безопасности организации, непротиворечиво включающей в себя все политики безопасности отделов, в случае объединения нескольких организаций в единую корпорацию необходимо дополнить каждую решетку, задающую мандатную политику безопасности для каждого из отделов, пустым элементом, после чего построить декартово произведение всех таких решеток. Это позволит получить непротиворечивую мандатную политику безопасности с отсутствием утечек информации.

Литература

1. Gyori I. *Some mathematical aspects of modelling cell population dynamics* // Computers and Math. Appl. 1990. V.20. N. 4. 6. P.127–138.
2. Гайдамакин Н.А. Разграничение доступа к информации в компьютерных системах. Уральский университет, 2003.
3. Биркгоф Г. Теория решеток / Г. Биркгоф. М.:Наука. Главная редакция физико-математической литературы, 1984. 568 с.

Математические структуры и моделирование
2010, вып. 21, с. 133–140

УДК 519.688

МЕТОД РАСПОЗНАВАНИЯ СПАМ-СООБЩЕНИЙ НА ОСНОВЕ АНАЛИЗА ЗАГОЛОВКА ПИСЬМА

А.Н. Мироненко

Разработана и реализована технология борьбы с массовыми рассылками на основе временной задержки сообщения и запроса о его повторной отправке. Проведено исследование влияния величины времени задержки сообщения на эффективность метода распознавания спам-сообщений.

Введение

Все современные методики борьбы со спамом можно условно разделить на следующие категории:

1. Методы, основанные на анализе письма. Их задача состоит в изучении письма с целью его классификации. Подобный метод решает две задачи: обнаруживает спам и выявляет письма, которые гарантированно не являются спамом. Известно несколько наиболее распространенных методов анализа:
   - По формальным признакам.
   - По содержимому с использованием сигнатурного анализа. Данный метод основан на поиске в тексте письма определенных сигнатур, описанных в обновляемой базе данных.
   - По содержимому с применением статистических методик. Большинство современных методов данного класса основано на теореме Байеса.
   - По содержимому с использованием SURBL (Spam URL Realtime Block Lists — списка блокировки спамерских URL). Идея метода состоит в поиске расположенных в теле письма ссылок и их проверке по базе SURBL. Этот метод эффективен против спама, в котором для обхода фильтров вместо рекламы применяется ссылка на сайт с рекламой.
2. Методы, основанные на признании отправителя письма в качестве спамера. Они опираются на различные «черные», «белые» списки IP- и почтовых адресов.

Омский государственный университет им. Ф.М. Достоевского.
E-mail: mironim84@mail.ru

3. Детекторы массовой рассылки. Как следует из названия, их задачей является обнаружение рассылки похожего письма большому количеству абонентов.
4. Методы, основанные на верификации обратного адреса отправителя и его домена. Простейшим методом защиты является обычный DNS-запрос по имени домена отправителя письма. Если выясняется, что домен отправителя не существует, то, вероятнее всего, адрес отправителя является поддельным. Однако этот метод малоэффективен, поскольку в качестве адреса отправителя спамеры могут использовать реальные адреса, случайным образом выбранные из базы рассылки.

Рассмотрим способ фильтрации спама, разработанный на основе анализа заголовка сообщения с запросом повтора отправки.

1. Описание разработанного метода фильтрации

Принцип, реализованный в данной работе, можно отнести сразу к нескольким категориям методик борьбы со спамом. Для распознавания спам-писем в нем используется комбинация проверки по «белому», «черному» списку и временной задержки сообщения. Данный метод обладает двумя ключевыми качествами, характеризующими эффективность работы любого метода фильтрации электронной почты. Это полнота и точность фильтрации. Под полнотой подразумевается процент обнаруженного спама, точность — это количество ложных срабатываний.

Пусть есть папка «карантин» (для задержки сообщений), список адресов электронной почты доверенных пользователей («белый» список), который на начальном этапе формируется самостоятельно, и «черный» список, изначально он пуст. Множество сообщений электронной почты (M). Время хранения сообщения в «карантине» (N) будет константой системы и выбирается на основе эксперимента. Первоначально при увеличении N эффективность метода возрастает, но при достижении некоторого порогового значения меняется слабо. В качестве рабочей величины N было выбрано 5 дней.

Первичная обработка входящего сообщения состоит в выделении заголовка письма. Он содержит следующие ключевые поля:

1. Return-Path — обратный адрес.

2. Received — строчка журналирования прохождения письма. Каждый почтовый сервер (MTA) помечает процесс обработки этим сообщением. Если сообщение проходит через несколько почтовых серверов, то новые сообщения дописываются над предыдущими (и журнал перемещения читается в обратном порядке, от ближайшего узла к самому дальнему).

3. From — имя и адрес отправителя. Может не совпадать с Return-Path.

4. Sender — отправитель письма. Добавлено для возможности указать, что письмо от чьего-то имени (from) отправлено другой персоной (например, секретарем от имени начальника).

5. To — имя и адрес получателя. Может содержаться несколько раз (если письмо адресовано нескольким получателям).

6. cc — (от англ. carbon copy) содержит имена и адреса вторичных получателей письма, к которым направляется копия.

7. bcc — (от англ. blind carbon copy) содержит имена и адреса получателей письма, чьи адреса не следует показывать другим получателям.

8. Reply-To — имя и адрес, куда следует адресовать ответы на это письмо.

9. Message-ID — уникальный идентификатор сообщения. Состоит из адреса узла-отправителя и номера (уникального в пределах узла). Алгоритм генерации уникального номера зависит от сервера. Выглядит примерно так: E1Nfsty-0001Ux-00. example-bk-ru@f118.mail.ru. Вместе с другими идентификаторами используется для поиска прохождения конкретного сообщения по журналам почтовой системы (почтовые системы фиксируют прохождение письма по его Message-ID) и для указания на письмо из других писем (используется для группировки и построения цепочек писем). Обычно создаётся первым почтовым сервером (MTA) в момент принятия почты от пользователя.

10. In-Reply-To — Указывает на Message-ID, для которого это письмо является ответом (с помощью этого почтовые клиенты могут легко выстраивать цепочку переписки — каждый новый ответ содержит Message-ID для предыдущего сообщения).

11. Subject —тема письма.

12. Date — дата написания письма.

Для данного метода распознавания спама важны поля Return-Path, Message-ID и Subject. Сообщение копируется по протоколу POP3. Затем проверяем вхождение адреса отправителя в «черный» список. Если он найден, то сообщение – спам, и оно удаляется, если нет, то проверяется по «белому» списку. Если адреса нет в списке, то сообщение копируется в папку «карантин» и определяется его Message-ID, указанный в заголовке. По шаблону формируется ответное сообщение, содержащее просьбу о подтверждении отправки с указанием в поле Subject ранее выявленного Message-ID. У каждого входящего сообщения Mi, попавшего в «карантин», адрес отправителя сравнивается с множеством адресов отправителей письма, от которых уже находятся в этой папке M-Mi. Если обнаруживается пара писем Mi и Mi+1 с одинаковыми адресами, то проверяется поле Subject письма Mi, содержащее некоторый набор символов, и сопоставляется с Message-ID Mi+1. В случае совпадения сообщение Mi удаляется, а Mi+1 перемещается в папку «входящие», и адрес отправителя добавляется в «белый» список. В случае не обнаружения пары M и M1, сообщение остается в «карантине»

и по истечении срока хранения удаляется с занесением адреса отправителя в «черный» список.

Если необходимо осуществить переписку с кем-либо без подтверждения отправки (добавив адресата в «белый» список автоматически), при составлении сообщения дописывается код в поле письма Subject, вся исходящая корреспонденция проверяется на наличие в ней этого кода. Если он обнаружен, то адрес автоматически добавляется в список доверенных, все ответы поступают без задержки.

2. Описание критериев оценки спам-фильтров

Для оценки качества работы антиспам-сервисов следует одновременно использовать два следующих критерия:

1. Точность (ложные срабатывания, или false positive) — доля нормальных (не являющихся спамом) сообщений, ошибочно классифицированных как спам в общем потоке нормальной почты.

2. Пропуск спама (false negative) — доля пропущенного спама в общем потоке спама. Полнота – доля отфильтрованного спама.

Обе характеристики нужно рассчитывать корректно, а именно:

1. Процент ложных срабатываний — это отношение числа нормальных писем, ошибочно признанных спамом, к количеству всей нормальной почты (пропущенных и заблокированных нормальных писем), а не от всего потока, включающего и спам тоже. Таким образом, 0,3% ложных тревог могут означать, например, что всего пришло 10 000 нормальных писем, и из них 30 было ошибочно признано спамом.

2. Процент пропусков — отношение количества пропущенного спама к объему всего спама (как пропущенного, так и распознанного). Таким образом, 15% пропусков (уровень фильтрации – 85%) означают, например, что всего пришло 10 000 спам писем, из которых 1 500 не было распознано как спам.

Ложные срабатывания можно разделить на критические и некритические.

В ситуациях, когда электронная почта является важным каналом коммуникации для компании, необходимо поддержание максимально низкой доли ложных срабатываний, особенно для важных деловых писем. Ущерб от потерянного делового письма может быть несопоставим с потерями рабочего времени от спама (это не означает, естественно, что спам вообще не нужно фильтровать).

При анализе ложных срабатываний недостаточно ограничиться только подсчетом их количества. В современных спам-фильтрах используются эвристические алгоритмы, которые могут распознать как спам (или «возможно спам») сообщения, «похожие на спам» (например, письмо всем пользователям интернет-магазина о скидках, написанное с использованием маркетинговой лексики), но при этом спамом с точки зрения получателей не являющееся. Целесообразно

при тестировании разделить ложные срабатывания на критические ложные срабатывания (ложные срабатывания на важной деловой или личной почте) и некритические (ошибочная классификация массовых новостных и маркетинговых рассылок и тому подобной почты) и подсчитывать процент тех и других отдельно.

Критерий на основе доли пропущенного спама является наиболее очевидным. Если один антиспам-фильтр распознает 70%, а второй — 85% спама, то второй фильтр можно считать лучшим. В то же время необходимо понимать, что повышение уровня распознавания может с большой вероятностью дать одновременный рост количества ложных срабатываний.

Поэтому оба критерия нужно рассматривать совместно, причем оценка количества ложных срабатываний должна иметь приоритет при составлении суммарной оценки фильтра.

3. Описание методик тестирования

Требования к эксперименту для получения наиболее достоверного результата следующие:

1. Реальная эксплуатация на реальном потоке почты в реальном времени. Наиболее достоверные результаты тестирования антиспам-систем можно получить только на реальном потоке почты и только при фильтрации немедленно, в реальном времени. Только в этом случае:
   - Распределение почты по типам (спам, неспам и так далее) соответствует реальному.
   - Техническая информация в письмах (IP-адрес посылающей стороны, SMTP envelope, технические заголовки) соответствует реальному положению дел. Содержимое баз данных фильтров (лингвистических, статистических, RBL-списков, «черных»,«белых» списков отправителей) является актуальным. Тексты писем не искажены за счет пересылки, вставки дополнительной информации или подобных действий.
   - Решается задача реальной фильтрации.
2. Тестирование должно продолжаться, как минимум, 2–3 недели. Поток, как спама, так и нормальной почты, сильно меняется во времени, обычно изменения тематики и оформления писем происходят ежедневно. Продолжительный тестовый период должен усреднить эти колебания. Полезно, если часть тестового периода может включить в себя сезонные изменения маркетинговой активности (предпраздничные распродажи, например). Это позволит оценить качество реакции антиспам-системы на пике спама.
3. При тестировании через систему должны пройти несколько десятков тысяч сообщений. В противном случае достоверно оценить уровень ложных срабатываний невозможно (так как приемлемый уровень некритических ложных срабатываний — не выше 0,01%, то есть одна ложная тревога на 10 тысяч писем или меньше).

4. В тестировании должны принимать участие, как минимум, несколько десятков почтовых ящиков. Это требование определяется тем, что вариативность потока спама у разных пользователей очень велика. Например, на ящики с именами info@, sales@ или alex@ приходит много мусорной почты, так как подобные имена легко подбираются методом словарной атаки, а на ящики со сложными именами наподобие Joe.V.User@ спама приходит во много раз меньше. Использование при тестировании большого числа почтовых ящиков позволяет усреднить вариации в потоках спама между различными типами почтовых ящиков.
5. Анализ результатов необходимо проводить с использованием единого определения спама и критичности, некритичности ложных срабатываний. Как пропуски спама, так и (в особенности) ложные срабатывания должны быть тщательно проанализированы. При оценке доли пропусков необходимо использовать корректное и единое для всех тестов определение спама. При оценке ложных срабатываний следует учитывать их критичность, поскольку это принципиально для оценки рисков использования конкретного фильтра.
6. Равные условия тестирования. При сравнении нескольких решений от разных производителей антиспам-фильтры должны быть поставлены в равные условия. Это включает в себя следующие требования:
   - Одинаковый поток почты, приходящий на разные фильтры в реальном времени.
   - При использовании RBL-сервисов — одинаковый набор списков RBL для всех тестируемых систем.
   - При использовании локальных «черных», «белых» списков — использование одинаковых списков.
   - При использовании обучаемых фильтров — обучение на одинаковых выборках. Если в процессе тестирования используется дообучение, дообучение должно быть синхронным по одним и тем же выборкам.
   - При использовании фильтров с получениями обновлений — синхронное получение обновлений.

Таким образом, достоверные результаты тестирования можно получить при выполнении следующих необходимых условий:

1. Тестирование в реальном окружении (установка антиспам-фильтра на тот же поток почты, где его предполагается в дальнейшем использовать) с достаточной продолжительностью тестирования — 2–3 недели.
2. Достаточный объем тестирующей выборки, как минимум, несколько тысяч сообщений в день.
3. Достаточная выборка почтовых ящиков, как минимум, несколько десятков.
4. Анализ результатов с использованием корректного определения спама и категорий критичных/некритичных ложных срабатываний.
5. Тестируемое ПО должно быть поставлено в максимально одинаковые условия.

4. Тестирование разработанного метода

Для корректного определения уровня качества фильтрации потока сообщений дадим определение понятию спам. Спам — это любые анонимные незапрошенные массовые рассылки электронной почты, как правило, имеющие рекламный характер.

На этапе подготовки к эксперименту по оценке эффективности метода было создано 12 ящиков электронной почты на разных сайтах: gmail.com, rambler.ru, mail.ru и yandex.ru. Для обеспечения потока почты на эти ящики они были использованы при регистрациях на различных сайтах, форумах за несколько недель до начала эксперимента. При этом на них была организована пересылка почты с почтового ящика, который используется для деловой переписки. Это сделано с целью обеспечения потока нормальных сообщений, что необходимо для оценки такого важного критерия эффективности метода фильтрации, как количество ложных срабатываний.

В среднем за 1 день на каждый ящик приходило около 25 писем (300 всего за 1 день), из них не спам – 5 (всего 60 сообщений за 1 день, 20% от общего числа), соответственно спам – 20 (240 всего за 1 день, 80% от общего числа). За все время эксперимента (32 дня) пришло 9 600 сообщений, из них не спам – 1920, соответственно спам – 7680. Изначально «белый» список содержал 2 адреса. «Черный» список был пуст. Время задержки сообщения в «карантине» 5 дней.

Программа, осуществляющая фильтрацию потока электронной почты, запускалась три раза в день: утром, днем и вечером. Перед запуском программы проверка содержимого ящиков производилась вручную, что позволило контролировать работу метода. За время тестирования метода за спам было принято 7 легитимных сообщений, что соответствует уровню ложных срабатываний – 0,07%. Так как общепринятого стандарта не существует, полученное значение можно считать допустимым при норме 0,01%. Допущенные ложные срабатывания не являлись критичными. Критичных ложных срабатываний не было допущено ни одного. Уровень фильтрации – 100%, это означает, что все 7680 спам-сообщения, пришедшие за время эксперимента, были отфильтрованы.

Выводы

В работе были рассмотрены существующие методы фильтрации потока электронной почты. Представлены методы и рекомендации по их тестированию.

В ходе работы был реализован и протестирован собственный метод фильтрации. Разработанный метод показал следующие результаты: полнота — 100%, весь спам был отфильтрован; точность, уровень некритичных ложных срабатываний – 0,07% (7 из 9 600), некритичных – 0%.

Таким образом, можно говорить, что разработанный метод может достаточно эффективно использоваться для фильтрации потока электронной почты.

Литература

1. Сергеев Д. «Черные» и «белые» списки как мера защиты от спама. URL: http://spam.knowledgebase.ru (дата обращения: 24.02.2010).
2. RFC 2076 (rfc2076) – Common Internet Message Headers. URL: http://www.faqs.org/rfcs/rfc2076.html (дата обращения: 20.02.2010).
3. Русских В. А сегодня вот – что почтальон и почта // Наука и жизнь. 1998. N. 3. URL: http://www.nkj.ru/archive/articles/10397/ (дата обращения: 20.02.2010).

Математические структуры и моделирование
2010, вып. 21, с. 141–150

УДК 004.056.57

ЧИСЛО ПОТОКОВ КАК ХАРАКТЕРИСТИКА ЗАРАЖЕННОСТИ ПРОЦЕССА

В.В. Пляшко, Н.Ф. Богаченко

В работе сформулированы и обоснованы критерии, позволяющие, опираясь на число потоков некоторого процесса, дать заключение о том, является ли этот процесс зараженным.

Введение

В рамках построения системы обнаружения вторжений, динамически настраивающейся под вновь обнаруживаемые уязвимости, актуальным становится ответ на вопрос: можно ли каким-то способом ничего не зная о структуре и логике процесса, по его свойствам определить, является ли он опасным.

Данная работа посвящена анализу возможности определения наличия заражения процесса вредоносными программами и срабатывания в процессе логической бомбы, опираясь на число потоков данного процесса. Результаты исследования можно представить в виде марковской цепи числа потоков, что находит отклик в современных подходах к построению систем обнаружения вторжений, использующих марковские модели [5, 7, 11].

Стоит отметить, что на сегодняшний день существует множество реализаций сканеров свойств запущенных процессов, например библиотеки `Psapi.dll` и `tlhelp32.dll`, а также конечные программные продукты, такие как «Стандартный диспетчер задач `Windows`», «`ProcessMonitor`» (`Procmon.exe`), «`ProcessExplorer`» (`procexp.exe`). Но ни один из них не предоставляет удобных инструментов для исследования возможности определения наличия вредоносных потоков в процессе.

1. Нежелательное изменение числа потоков

В этом разделе речь пойдет о том, в каких случаях можно говорить о нежелательном изменении числа потоков исследуемого процесса [10].

Во-первых, вспомним о так называемой логической бомбе, которая может содержаться даже в самых, казалось бы, безобидных приложениях. При этом

Омский государственный университет им. Ф.М. Достоевского.
E-mail: `nfbogachenko@mail.ru`

она может быть как намеренно «вшита» в программу, так и являться результатом работы так называемого EPO-вируса [2].

Логическая бомба — это программа или часть программы, которая запускается при определенных временных или информационных условиях для осуществления вредоносных действий (как правило, несанкционированного доступа к информации, искажения или уничтожения данных). Многие вредоносные программы, такие как вирусы или черви, часто содержат логические бомбы, которые срабатывают в заранее определенное время или при выполнении определенных условий, например в пятницу 13-го. К логическим бомбам, как правило, относят код, который приводит к не сообщенным заранее последствиям для пользователей. Активировать логическую бомбу может и сам пользователь, как умышленно, так и неумышленно, даже не подозревая об этом.

Будем рассматривать запускаемый процесс как совокупность его состояний. Условно разделим все состояния процесса на нормальные (не несущие угроз безопасности) и нежелательные. На рисунке 1 верхней стрелкой обозначено нормальное функционирование процесса, левой – срабатывание логической бомбы или иной способ перехода процесса в нежелательное состояние. Последнее может быть реализовано каким угодно образом, в том числе процесс в этом состоянии может запрашивать дополнительные ресурсы, например открывать новые потоки.

Рис. 1. Общая модель работы процесса

Стоит заметить, что срабатывание логической бомбы может повлечь за собой достаточно серьезные последствия. Ярким свидетельством этого является опыт войны в Персидском заливе: Ирак не смог применить против многонациональных сил закупленные во Франции системы ПВО потому, что их программное обеспечение содержало логические бомбы, которые были активизированы с началом боевых действий [9].

Рассмотрим теперь другой случай. Допустим, что у самого процесса нет вышеописанных нежелательных состояний. Здесь имеет место другой фактор: при написании вредоносных программ злоумышленники часто используют достаточно популярный способ скрыть существование своей программы, замаскировав ее выполнение под поток другого процесса. Схематично это показано на рисунке 2.

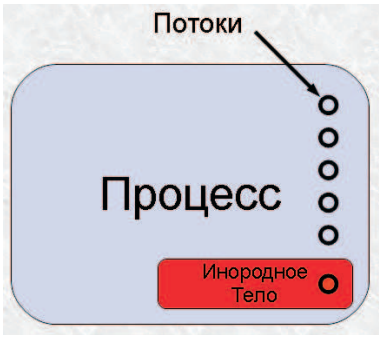


Рис. 2. Схема заражения процесса

Например, это можно сделать следующим образом: зарегистрировать фрагмент памяти, в котором будет лежать код (для этого можно воспользоваться функциями `OpenFileMapping` и `MapViewOfFile` из `kernel32.dll`). В эту область поместить код потока, который будет выполняться в процессе-жертве. Теперь необходимо заставить жертву запустить этот поток. Для этого можно использовать функцию `CreateRemoteThread` [6]. Есть и другие описания подобных механизмов, например в работе [4].

Оба рассмотренных случая демонстрируют, что нежелательное поведение процесса, будь то его собственные функции или внешнее вмешательство, может привести к увеличению числа потоков. Следовательно, есть теоретическая возможность узнать об угрозе безопасности, опираясь на число потоков приложения.

2. Этапы эксперимента и необходимые ограничения

Изучение поведения потоков исследуемого процесса предлагается разбить на два этапа:

– этап наблюдения: изучение поведения в идеальных условиях – *режим обучения*;

– этап аудита: анализ поведения в естественных условиях – *рабочий режим*.

При этом требуется понять, какие ограничения накладываются на планируемый эксперимент. Сразу стоит заметить, что исследование нужно проводить в общем виде, соответственно мы ничего не можем знать о структуре и логике процесса, который нам предстоит изучать. В связи с этим мы априори не сможем отличить нежелательные состояния от нормальных.

Чтобы научиться отличать друг от друга данные состояния, необходимо определить, что такое нормальные состояния процесса.

1. Во время наблюдения за процессом считается, что угрозы безопасности отсутствуют.

Это условие гарантирует, что любое состояние, замеченное во время наблюдения, является нормальным. Но не стоит путать состояние процесса в общем смысле и состояние процесса в смысле количества его потоков, ибо при одном и том же числе потоков процесс может выполнять или не выполнять совер-

шенно разные наборы функций, в том числе и вредоносные. У нас же имеется информация только о числе потоков. Обозначим его ν.

Итак, мы наблюдаем за процессом, собирая информацию обо всех возможных значениях величины ν. Как правило, для сложных приложений возможность получить весь набор значений величины ν очень сомнительна. Поэтому примем следующее допущение:

2. Либо мы имеем достаточно времени и возможностей для получения всех значений величины ν нормального функционирования процесса, либо все значения величины ν, которые не были получены во время наблюдения, являются признаком нежелательного состояния процесса.

3. Параметры исследования

В дальнейшем под *состоянием процесса* будем понимать текущее число потоков этого процесса, то есть значение величины ν. Тогда *разрешенное состояние* – это то, в котором был замечен процесс в режиме обучения. *Запрещенное состояние* – это то, которое не было замечено при обучении.

Опираясь на допущения предыдущего раздела, можно говорить, что *в режиме обучения мы получаем все возможные разрешенные состояния.*

Безусловно, мы сможем определить наличие вредоносных потоков у приложений с постоянным числом потоков, как в принципе и у приложений, имеющих небольшое ограниченное число возможных потоков.

Можно привести простой пример: утилиту `tracert`, которая работает в один поток. Соответственно, если у нее возникает один лишний поток, сразу понятно, что ее поведение не является безопасным.

Другой пример – программа `mspaint`. На этапе обучения становится понятно, что данная программа запускает в начале пять потоков. При вызове меню «Открыть файл» количество потоков увеличивается на три, позже постепенно уменьшаясь. Таким образом, мы можем выделить характерные изменения числа потоков: «+3» или «–1». Для наглядности функция изменения числа потоков программы изображена на рисунке 3 в виде направленного графа [1]. Заметим,

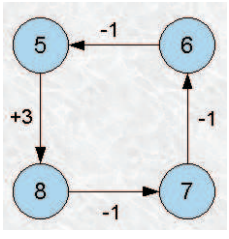


Рис. 3. Возможные изменения числа потоков

что увеличение числа потоков на величину, отличную от трех, в данном случае является неестественным и может свидетельствовать о том, что процесс небезопасен. Следовательно, *возможные изменения числа потоков (изменения величины ν) могут оказаться важной характеристикой.*

Теперь попробуем добавить к исследованиям еще одну функцию приложения `mspaint` – открытие справки. Если в режиме обучения открывать диалог «Открыть файл» и справку в произвольном порядке, то у нас получится более разнообразный набор состояний и переходов. Так, открытие справки влечет за собой прибавку 6-ти потоков. Закрытие же ее приводит к немедленному уменьшению числа потоков на четыре и последующему постепенному уменьшению числа потоков еще на два.

Рассмотрим рисунок 4. Как мы видим, допустимо увеличение числа потоков

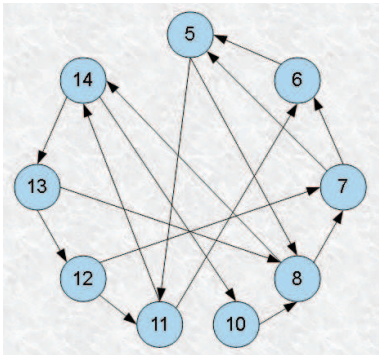


Рис. 4. Возможные изменения числа потоков при более длительном наблюдении

на три. Но из всех ли состояний возможно данное увеличение? Естественно, что заведомо вызовет подозрения, если увеличение на три произойдет с 12-ти, 13-ти или 14-ти потоков, так как итоговое количество не будет даже входить в набор, полученный в режиме обучения.

Нас больше интересуют такие изменения числа потоков на три, при которых мы из разрешенного числа потоков попадаем опять же в разрешенное. Примером может служить несуществующий переход от 7-ми потоков к 10-ти. Это означает следующее: *недостаточно учитывать допустимые изменения числа потоков (изменения величины ν), а следует также фиксировать состояния, из которых возможны данные изменения.*

Необходимость учета таких *исходных* состояний подтверждает и следующий факт. Один и тот же исполняемый файл при разных начальных параметрах может привести к различному поведению соответствующего процесса. Пример возможной модели функционирования процессов, запущенных из того же исполняемого файла, но с разными параметрами, представлен на рисунке 5: изменения числа потоков «+1» и «–1» характерны для обоих вариантов, «+2» и «–2» – только для первого, а «+5» – только для второго.

Теперь представим, что исполняемый файл исследуемого процесса после завершения обучения был инфицирован. И допустим, что пользователю не понадобятся функции открытия файла и справки. В этом случае возможен вариант, что вредоносная программа выделяет для себя необходимые ресурсы, запуская часть себя в отдельном потоке, после чего передает управление своей «жертве». В этом случае получится процесс со стартовым количеством потоков «6», что является разрешенным числом. Но факт того, что файл был заражен, усколь-

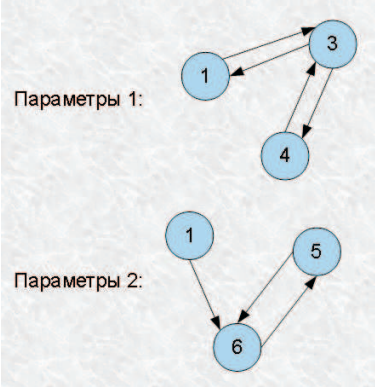


Рис. 5. Поведение процесса при различных начальных параметрах

зает. Чтобы этого не произошло, можно также проверять *стартовое число потоков процесса*.

4. Поиск критерия

Пусть на этапе обучения собрана некоторая информация о динамике изменения числа потоков исследуемого процесса, назовем ее статистикой.

Не будем забывать, что поведение процессов может существенно отличаться, и некоторые из них являются более предсказуемыми в плане определения наличия вредоносных потоков, а некоторые могут быть очень «неудобными» в этом смысле. В связи с этим было бы целесообразно найти такой критерий, который, исходя из собранной информации, мог бы давать нам представление о том, насколько каждый конкретный процесс является *предсказуемым* в вопросе определения наличия вредоносных потоков.

Естественно, что одного математического ожидания числа потоков будет явно недостаточно. Проанализируем, насколько подходит дисперсия на роль критерия [3, 8].

Рассмотрим процесс с постоянным числом потоков. Очевидно, что математическое ожидание числа потоков равно самому числу потоков, а дисперсия числа потоков такого процесса будет равна нулю:

$$M_1 = \text{const}, \ D_1 = 0.$$

Рассмотрим процесс с двумя равновероятными состояниями: $\nu_1 = 1, p_1 = 0.5$, $\nu_2 = 2, p_2 = 0.5$. Математическое ожидание и дисперсия числа потоков:

$$M_2(\nu) = 1.5, \ D_2(\nu) = 0.25.$$

Теперь предположим, что есть три равновероятных состояния: $\nu_1 = 1$, $p_1 = 1/3$, $\nu_2 = 2$, $p_2 = 1/3$, $\nu_3 = 3$, $p_3 = 1/3$. Математическое ожидание и дисперсия числа потоков:

$$M_3(\nu) = 2, \ D_3(\nu) = 0.667.$$

Нетрудно заметить, что в приведенных примерах с ростом возможного числа потоков при равной вероятности их появления дисперсия возрастает: $D_1 < D_2 < D_3$. Можно предположить, что чем больше дисперсия, тем более разнообразно ведет себя процесс и тем сложнее определить, присутствует ли в нем вредоносный поток. Попробуем найти пример, противоречащий данному предположению: $\nu_1 = 2$, $p_1 = 0.5$, $\nu_2 = 6$, $p_2 = 0.5$. Математическое ожидание и дисперсия числа потоков:

$$M_4(\nu) = 4,\ D_4(\nu) = 4.$$

С одной стороны, выполняется следующая цепочка неравенств: $D_1 < D_2 < D_4$. С другой – как в первом, так и в четвертом примерах прибавка одного вредоносного потока приводит к запрещенному состоянию, что автоматически указывает на нежелательное состояние процесса. Во втором примере та же самая прибавка одного вредоносного потока может привести к переходу из разрешенного состояния опять же в разрешенное.

Таким образом, дисперсия не подходит для оценки возможности определения наличия вредоносных потоков. Необходимо найти подход, учитывающий не только разброс возможного числа потоков, но и сам спектр принимаемых значений.

Поместим все разрешенные состояния процесса на числовую прямую, например так, как показано на рисунке 6. При фиксированных минимальном и

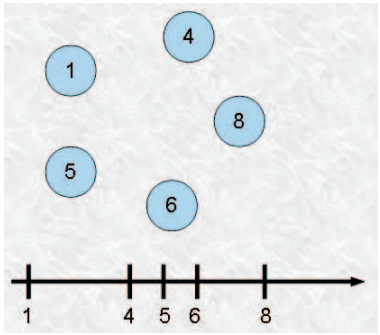


Рис. 6. Размещение значений величины ν на числовой прямой

максимальном значениях количество гипотетически возможных целочисленных значений величины ν известно. В данном случае – 8. Так же возможно подсчитать число разрешенных значений – 5. Очевидно, что *чем больше в данном промежутке разрешенных состояний и чем меньше запрещенных, тем меньше шансов обнаружить нежелательное увеличение числа потоков.*

Рассмотрим два примера, показанных на рисунке 7: в обоих случаях длина промежутка и количество разрешенных значений величины ν совпадают. Но из-за того, что во втором примере разрешенные состояния располагаются «кучно», будет сложнее определить наличие, скажем, одного вредоносного потока. В то время как первый пример позволяет сделать это почти во всех разрешенных состояниях.

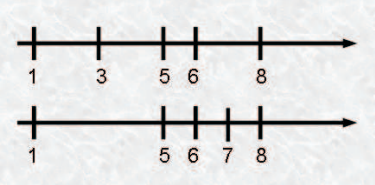


Рис. 7. Разрешенные состояния процессов

Таким образом, *имеет смысл учитывать не только спектр принимаемых параметром значений, а также особенности их взаимного размещения.*

Помимо вышесказанного стоит также учитывать специфику возможного нежелательного поведения процесса. Хотя вредоносные средства и пишутся таким образом, чтобы быть менее заметными в системе и использовать как можно меньше ресурсов, но они могут быть реализованы не только в одном, но и в нескольких потоках.

Обозначим количество вредоносных потоков через X. Теперь мы можем сформулировать так называемый *критерий X-непредсказуемости процесса.* Рассмотрим пример, изображенный на рисунке 8. Для начала определим, на-

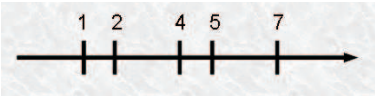


Рис. 8. Разрешенные состояния процесса

сколько удобен процесс для определения наличия одного вредоносного потока. Для этого разделим все его разрешенные состояния на 1-предсказуемые и 1-непредсказуемые. *1-предсказуемыми* назовем те состояния, увеличение числа потоков на один в которых приведет к запрещенному состоянию. Соответственно *1-непредсказуемыми* состояниями назовем те, увеличение числа потоков на один в которых приведет к разрешенному состоянию. На рисунке 8 состояния «1» и «4» являются 1-непредсказуемыми.

Изучение поведения процесса на этапе обучения позволяет вычислить вероятности[1] нахождения процесса в каждом состоянии. Теперь мы можем говорить о вероятности нахождения процесса в 1-непредсказуемом состоянии в каждый момент времени: для этого нужно сложить вероятности нахождения процесса в каждом из 1-непредсказуемых состояний. Найденная вероятность и будет значением критерия 1-непредсказуемости.

По аналогии можно вычислить значения критерия для двух и более вредоносных потоков: 2-непредсказуемыми являются состояния «2» и «5»; 3-непредсказуемыми – состояния «1», «2» и «4»; 4-непредсказуемое состояние «1» и т.д. Для расчета критерия X-непредсказуемости в каждом случае надо сложить вероятности X-непредсказуемых состояний.

[1]Здесь и далее речь пойдет о вероятностях, экспериментально полученных в ходе обучения.

5. Разработка программы

Для проведения численного эксперимента на языке `C++` [13] в среде разработки `Microsoft Visual Studio` [14] была реализована программа-сканер, отслеживающая число потоков запущенных процессов.

Предусмотрено три режима работы сканера – простой, режим обучения и рабочий режим. Программа-сканер сохраняет собранную на этапе обучения статистику в файл. Для сохранения данных был выбран формат XML [12]. Имеется функция обнуления накопленной статистики и возможность отключать слежение для каждого конкретного процесса.

Для удобства на этапе обучения для каждого процесса сохраняется не величина изменения числа потоков, а состояние, в которое процесс попадает при таком изменении (см. таблицу 1). Фиксируется количество попаданий процесса

Таблица 1. Описание разрешенного поведения процесса, представленного на рисунке 5

Разрешенное состояние	1	3	4	5	6
Разрешенные состояния перехода	3, 6	1, 4	3	6	5

в каждое состояние и количество переходов из одного допустимого состояния в другое. Далее происходит расчет математического ожидания и дисперсии числа потоков, и вычисляется значение критерия 1-непредсказуемости процесса.

В представленной работе подробно освещены алгоритмы режима обучения. Стоит заметить, что расчет критерия X-непредсказуемости пока что не учитывает условные вероятности перехода из одного состояния в другое – это вопрос дальнейших исследований.

Накопленная в ходе обучения статистика в рабочем режиме используется для диагностики процессов. Описание алгоритмов этого режима выходит за рамки данной статьи.

В заключении построим граф состояний некоторого процесса (см. рис. 9): здесь вес ребра – полученное на этапе обучения число соответствующих пере-

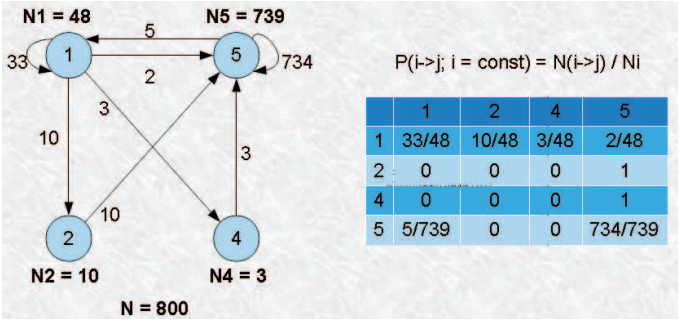


Рис. 9. Граф состояний и матрица условных вероятностей перехода

ходов. Используя количественные показатели, вычисляются условные вероятности перехода из одной вершины в другую. Таким образом, можно говорить о

марковской цепи с дискретным временем, матрица которой также представлена на рисунке 9.

Не стоит забывать, что программа написана для работы в пользовательском режиме. Если же иметь доступ к таблице прерываний, можно точно определять время запуска потоков, и тогда речь уже пойдет о марковской цепи с непрерывным временем. Это может дать дополнительные возможности исследования поведения процессов.

Литература

1. Берж К. Теория графов и ее приложения. М.: ИЛ, 1962. С. 11-14.
2. Бойцев О. Защити свой компьютер на 100 % от вирусов и хакеров. СПб.: Питер, 2004. С. 98.
3. Гмурман В.Е. Теория вероятностей и математическая статистика. М.: «Высшая школа», 2003. С. 85-89.
4. Зайцев О. Технологии вредоносных программ и угрозы информационной безопасности // КомпьютерПресс. 2007. N. 3.
   URL: http://www.compress.ru/article.aspx?id=17361 (дата обращения: 10.01.2010).
5. Ивашко Е.Е. Построение системы защиты электронных библиотек от несанкционированного копирования документов.
   URL: http://rcdl2007.pereslavl.ru/papers/paper_41_v1.pdf (дата обращения: 27.03.2010).
6. Касатекно И. Пишем свой стелс // Хакер. 2003. N. 35(10). С. 52.
7. Кашаев Т.Р. Система активного аудита на основе скрытых марковских моделей // Информационное противодействие угрозам терроризма. Научно-практический журнал.
   URL: http://www.contrterror.tsure.ru/site/magazine4/Pdf/Journal4full.pdf (дата обращения: 27.03.2010).
8. Кибзун А.И. Теория вероятностей и математическая статистика. М.: Физматлит, 2002. С. 58.
9. Поздняков А.И. Информационная безопасность личности, общества, государства // Военная мысль. 1993. С. 16.
10. Рихтер Д. Windows для профессионалов. Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. СПб.: Питер, 2004. С. 130-154.
11. Соколов А.М. Обнаружение вторжений в компьютерную систему с помощью марковских цепей переменного порядка.
    URL: http://www.iai.dn.ua/public/JournalAI_2002_4/Razdel1/11_Sokolov.pdf (дата обращения: 27.03.2010).
12. Спенсер П. XML. Проектирование и реализация. М.: Лори, 2001.
13. Хортон А. Visual C++ 2005 Базовый курс. М.: Диалектика, 2007.
14. Kruglinski D.J., Wingo S., Shepherd G. Programming Microsoft Visual 6.0. Microsoft Press, 1998.

Математические структуры и моделирование
2010, вып. 21, с. 151–161

УДК 65.012.810

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД В ПОСТРОЕНИИ ПОЛИТИКИ БЕЗОПАСНОСТИ. СИСТЕМЫ С ЕСТЕСТВЕННОЙ ИЕРАРХИЕЙ

С.В. Усов

Проведена модификация модели дискреционного разделения доступа HRU для объектно-ориентированных компьютерных систем с учетом иерархии классов объектов.

Введение

Традиционно рассматриваемая модель системы безопасности компьютерных систем строится на основе субъектно-объектного подхода. Однако в последнее время, в связи с распространением объектно-ориентированного подхода в построении программного обеспечения компьютерных систем, складывается двойственная ситуация, когда один и тот же набор данных в одном случае интерпретируется как объект (пассивный), в другом случае как субъект (активный). В связи с этим, для более адекватного описания компьютерных систем, необходимо применение объектно-ориентированного подхода и соответствующая модификация моделей политик безопасности. В частности, объектно-ориентированная модель компьютерной системы построена в работе [1], где даны и необходимые определения. В последующей работе [2] определены элементарные операторы в рамках модели безопасности HRU [3] для объектно-ориентированных систем, а также установлена разрешимость проблемы безопасности системы для отдельных случаев, а именно для монооперационных HRU-систем и моноусловных монотонных HRU-систем.

В настоящей работе мы рассмотрим некоторые частные случаи моделей безопасности в объектно-ориентированных системах.

1. Классификация моделей безопасности объектно-ориентированных систем

Прежде всего определим HRU-модель, рассмотренную в работе [2], как *свободную*, т.к. в ней не накладывается никаких дополнительных ограничений на

Омский государственный университет им. Ф.М. Достоевского.
E-mail: raintower@mail.ru

отношения между уровнями доступа различных объектов различных классов. Однако наряду с HRU-подходом в сфере защиты информации распространен и иерархический подход [4], когда на множестве O объектов задан частичный порядок по уровню доступа.

Будем рассматривать компьютерную систему в виде множества объектов классов K, имеющих открытые поля f и скрытые поля p, а также методы обработки полей s. Пусть $O^k \subset O$ – множество объектов класса $k \in K$. В случае, если требуется уточнить класс объекта, поле f объекта $o^k \in O^k$ будем обозначать $o^k.f$, поле f класса k – $k.f$. Для скрытых полей и методов класса будем использовать аналогичные обозначения.

Все необходимые определения даны в работе [1], там же построена и модель системы безопасности для объектно-ориентированной компьютерной системы.

Матрица доступа M в данном случае - скрытое (private) поле объекта. От объекта к объекту даже одного класса (типа) содержание этого поля может отличаться. Строки матрицы объекта $o \in O$ соответствуют объектам компьютерной системы, столбцы - открытым (public) полям и методам объекта o, элементом матрицы является некоторое подмножество множества R видов доступа к полю, если столбец элемента соответствует public полю объекта и принимает значение 0 либо 1, если столбец элемента соответствует методу объекта, где «1» разрешает вызов метода, а «0» – запрещает.

Определение 1. HRU-модель системы безопасности называется иерархической, если на множестве объектов O задан частичный порядок-иерархия и в любой момент работы системы для любых двух объектов o, o' таких, что $o' \leq o$, для любого поля или метода $x \in X$, общего для объектов o и o', и для любого поля или метода $y \in X$ объекта o'' верно следующее: $o''.M[o, o''.x] \subset o''.M[o', o''.x]$ и $o'.M[o'', o'.x] \subset o.M[o'', o.x]$. Здесь и далее X - множество всевозможных полей и методов всех существующих в системе на данный момент времени объектов, " $\leq$ " – отношение частичного порядка.

Иерархическую систему можно представить в виде графа с вершинами, помеченными классами, и ребрами, задающими отношение частичного порядка.

Частным случаем является система на множестве объектов, на котором задано отношение полного порядка.

Однако в объектно-ориентированных системах классы (и, соответственно, объекты этих классов) уже связаны частичным отношением наследственности, поэтому в данном случае иерархия строится естественным образом.

Определение 2. HRU-модель объектно-ориентированной системы безопасности называется моделью с естественной иерархией, если в любой момент работы системы для любых двух объектов o^k, $o^{k'}$ классов k и k' таких, что k' является потомком k, для любого поля или метода x, общего для классов k и k', и для любого поля или метода $y \in X$ объекта $o^{k''}$ выполняются следующие соотношения: $o^{k''}.M[o^k, o^{k''}.x] \subset o^{k''}.M[o^{k'}, o^{k''}.x]$ и $o^{k'}.M[o^{k''}, o^{k'}.x] \subset o^k.M[o^{k''}, o^k.x]$. Здесь X - множество всевозможных полей и методов всех существующих в системе на данный момент времени объектов.

В графе системы с естественной иерархией ребра задают отношение наследственности на множестве классов и направлены от родителей к непосредственным потомкам, также имеется выделенная вершина-корень - это пустой класс-прародитель, которым прямо или опосредованно порождены все остальные классы. Класс будем называть финальным, если он не имеет наследников.

Определение 3. Класс объектно-ориентированной системы безопасности называется однородной, если матрицы доступа всех объектов этого класса в любой момент времени совпадают. В этом случае имеет смысл рассматривать матрицу доступа как атрибут не отдельного объекта класса, а класса целиком.

Определение 4. HRU-модель объектно-ориентированной системы безопасности называется однородной, все ее классы однородны.

Определение 5. HRU-модель объектно-ориентированной системы безопасности с естественной иерархией называется ациклической, если граф иерархии ацикличен.

Заметим, что модели с естественной иерархией либо однородны, либо все объекты системы являются представителями классов, не имеющих наследников. Во втором случае матрица доступов объекта o класса k подчиняется соотношениям из определения естественной иерархии как матрица наследника по отношению к матрице самого класса k.

Также отметим, что в работе [2] были рассмотрены свободные неоднородные HRU-модели объектно-ориентированных систем.

2. Однородные объектно-ориентированные системы с естественной иерархией

Определим следующие элементарные операции для работы с матрицей доступов:

1. $Create(o, k)$ – создает объект o класса $k \in K$, при этом $O := O \cup \{o\}$.

2. $Destroy(o, k)$ – уничтожает объект o, при этом $O := O \setminus \{o\}$.

Первые две операции никак не отражаются на матрице доступов.

3. $Enter(r, k, k'.f)$ – вносит право доступа r в $k'.M[k, k'.f]$, при этом матрица доступа изменяется по правилу: $k'.M[k, k'.f] := k'.M[k, k'.f] \cup \{r\}$. Данная элементарная операция может быть выполнена только при следующих условиях:

а) для любого потомка k_1 класса k, $r \in k'.M[k_1, k'.f]$;

б) для любого родителя k_2 класса k', обладающего полем f, $r \in k_2.M[k, k_2.f]$.

4. $Delete(r, k, k'.f)$ – удаляет право r доступа из $k'.M[k, k'.f]$, при этом матрица доступа изменяется по правилу: $k'.M[k, k'.f] := k'.M[k, k'.f] \setminus \{r\}$. Данная элементарная операция может быть выполнена только при следующих условиях:

а) для любого родителя k_1 класса k, обладающего полем f, $r \notin k'.M[k_1, k'.f]$;

б) для любого потомка k_2 класса k', $r \notin k_2.M[k, k_2.f]$.

5. $Grant(k, k'.s)$ – разрешает вызов объекту класса k метода $k'.s$, при этом матрица доступа изменяется по правилу $k'.M[k, k'.s] := 1$. Данная элементарная операция может быть выполнена только при следующих условиях:

а) для любого потомка k_1 класса k $k'.M[k_1, k'.s] = 1$;

б) для любого родителя k_2 класса k', обладающего методом s, $k_2.M[k, k_2.s] = 1$.

6. $Deprive(k, k'.s)$ – запрещает вызов объекту класса k метода $k'.s$, при этом матрица доступа изменяется по правилу $k'.M[k, k'.s] := 0$. Данная элементарная операция может быть выполнена, только при выполнении следующих условий:

а) для любого родителя k_1 класса k, обладающего методом s; $k'.M[k_1, k'.s] = 0$

б) для любого потомка k_2 класса k' $k_2.M[k, k_2.s] = 0$.

Условия а) и б) выполнения операций $Enter$, $Delete$, $Grant$, $Deprive$ будем называть условиями целостности.

Без ограничения общности можно считать, что не обладающий полем (или методом) $x \in X$ класс все же таковым обладает, но это поле пусто (или метод ничего не выполняет), а доступом к такому полю (методу) обладают все классы.

Определение 6. Под состоянием компьютерной системы в момент времени t будем понимать пару $Q(t) = (O(t), M(t))$, где $O(t) = \{o_j\}$ - множество всех объектов системы в момент времени t, а $M(t) = \{k.M[](t), k \in K\}$ - семейство всех матриц доступа классов системы в состоянии на момент t. Начальное состояние системы будем обозначать $Q(0)$.

Состояния компьютерной системы в модели HRU изменяются под воздействием запросов на модификацию матрицы доступа в виде команд $\gamma(x_1 : k_1, \ldots, x_m : k_m)$ следующего формата:

if – условная часть (представляет собой конъюнкцию значений логических выражений вида $r \in k'.M[k, k'.f]$ или $k'.M[k, k'.s] = 1$);

$then$ – последовательность элементарных операций.

Здесь аргументы x_i, $i = 1 \ldots m$ представляют собой поля или функции классов $k_i \in K$, участвующие в качестве аргументов в условной части либо в элементарных операциях. Классы аргументами команды не являются: они определяют саму команду.

Отметим, что помимо уже входящих в команду условий каждая операция $Enter$, $Delete$, $Grant$, $Deprive$ по умолчанию добавляет в if-часть команды свои условия целостности, соединенные операцией конъюнкции. Таким образом, если хотя бы одно условие целостности не будет соблюдено, вся команда не будет выполнена.

Кроме того, не должно существовать команд, содержащих одновременно операции $Enter(r, k, k''.f)$ и $Delete(r, k, k'.f)$, либо $Grant(k, k''.s)$ и $Deprive(k, k'.s)$, либо $Enter(r, k', k.f)$ и $Delete(r, k'', k.f)$, либо $Grant(k', k.s)$ и $Deprive(k'', k.s)$, где k'' - потомок k'. Несоблюдение этого условия приведет к нарушению естественной иерархии. Действительно, пусть, к примеру, $Enter(r, k', k.f)$ и $Delete(r, k'', k.f)$ присутствуют в одной команде, и в момент, когда $r \in k.M[k'', k.f]$, но $r \notin k.M[k', k.f]$, команда выполняется, в результате

чего класс k'' (потомок) теряет право r на поле $k.f$, а родительский класс k' его приобретает.

Все прочие сочетания элементарных операций являются допустимыми, т.к. не могут привести к нарушению естественной иерархии.

Наконец, обратимся к вопросу безопасности однородных систем с естественной иерархией.

Факт перехода системы под действием команды γ из состояния $Q(t)$, в котором она находилась в момент времени t, в состояние $Q(t+1)$, в котором она находилась в следующий момент времени $t+1$, будем записывать в сокращенном виде: $Q(t) \to_\gamma Q(t+1)$.

Определение 7. Будем говорить, что состояние однородной системы $Q(t)$ допускает утечку права доступа $r \in R$, если существуют такие команда γ, класс k и поле $k'.f$ класса k', что $r \notin k'.M[k, k'.f](t)$, но $r \in k'.M[k, k'.f](t+1)$, где $Q(t) \to_\gamma Q(t+1)$.

Определение 8. Будем говорить, что состояние однородной системы $Q(t)$ допускает утечку права вызова, если существуют такие команда γ, класс k и метод $k'.s$ класса k', что $k'.M[k, k'.s](t) = 0$, но $k'.M[k, k's](t+1) = 1$, где $Q(t) \to_\gamma Q(t+1)$.

Определение 9. Будем говорить, что система r-безопасна, если не существует такой последовательность команд $\gamma_1 \dots gamma_T$, что $Q(t-1) \to_{\gamma_t} Q(t)$, $t = 1, 2 \dots T$, и $Q(T)$ допускает утечку права r либо утечку права вызова.

Оказывается, проблема безопасности однородных систем решается очень просто.

Теорема 1. *Проблема r-безопасности системы является NP-разрешимой для однородной объектно-ориентированной модели (в том числе, и для модели с естественной иерархией).*

Доказательство. Мы исходим из предположения, что рассматриваемая система не является динамической относительно своей классовой структуры, в частности, количество классов постоянно и их структура определена заранее. В таком случае конечным является и множество состояний всех матриц доступа этих классов (т.к. множество прав доступа R конечно, и множество классов K конечно). А значит, конечно и количество операций, изменяющих состояния ячеек матрицы доступа. Наконец, отсюда следует, что конечно количество различных команд, которые можно составить из этих операций.

Будем искать цепочку команд наименьшей длины, переводящую систему в состояние, допускающее утечку некоторого права r. Заметим, что такая цепочка не должна содержать команд, не изменяющих ни одну матрицу доступа (как, например, команда, добавляющая право r в ячейку матрицы доступа, в которой право r уже имеется).

Допустим,

$$Q(0) \to_{\gamma_1} Q(1) \to \dots \to_{\gamma_{T-1}} Q(T-1) \to_{\gamma_T} Q(T),$$

где $Q(T)$ допускает утечку права $r \in R$ в ячейку $k.M[k', k.f](T)$.

В такой цепочке одна и та же команда не может выполняться после одинаковых состояний системы: действительно, пусть в цепочке присутствует фрагмент

$$Q(t_1) \to_\gamma Q(t_2)$$

и фрагмент

$$Q(t_3) \to_\gamma Q(t_4),$$

причем

$$Q(t_1) = Q(t_3),$$

но тогда

$$Q(t_2) = Q(t_4),$$

и цепочку можно сократить:

$$Q(t_1) \to_\gamma Q(t_4).$$

Таким образом, ни одна команда не встречается в цепочке большее число раз, чем количество всевозможных состояний системы.

Итак, длина минимальной цепочки, допускающей утечку права, конечна, и количество всевозможных команд в цепочке конечно. Получаем, что нам достаточно перебрать конечное множество цепочек. ■

3. Неоднородные объектно-ориентированные системы с естественной иерархией

Во-первых, отметим, что такие системы довольно сложны в реализации. Действительно, все объекты произвольного класса k обладают (по включению) не меньшим набором прав, чем любой объект каждого класса-родителя k, и не большим, чем любой объект каждого класса-наследника k. С другой стороны, на произвольный метод или поле x каждого объекта класса k предъявляется (по включению) прав не меньше, чем на соответствующий метод любого объекта класса-наследника k, и не больше, чем на соответствующий метод любого объекта класса-родителя k. Причем набор прав доступа любого объекта к любому полю должен изменяться в процессе работы системы без нарушения вышеназванных ограничений.

Во-вторых, в реальных системах, как правило, происходит работа только с объектами финальных классов, а вспомогательные (не являющиеся финальными) классы из цепи, соединяющей прародителя и финальный класс (т.е. все остальные классы этой цепи), не используются, либо объекты этих классов неотличимы в смысле набора прав доступа, т.е. классы являются однородными. В таких системах матрицы доступа объектов нефинального (а значит, однородного) класса будем отождествлять с матрицей доступа самого класса, а набор прав доступа финального класса будем искать как нижнюю грань (по включению) по всем наборам прав объектов этого класса. Таким образом, матрицы доступа на классах можно считать изначально заданными.

Определение 10. Неоднородные объектно-ориентированные системы с естественной иерархией будем называть финально-неоднородными, если все ее нефинальные классы однородны, и для любого финального класса k, его объекта o^k, его поля или метода x и произвольного объекта o', содержащего поле или метод x', выполнено: $o^k.M[o', o^k.x] \subset k.M[o', k.x]$ и $o'.M[k, o'.x'] \subset o'.M[o^k, o'.x']$.

Заметим, что неоднородная система сводится к финально-неоднородной добавлением к каждому неоднородному классу k наследника k', совпадающего со своим родителем во всем, вплоть до прав доступа. Такой наследник будет являться финальным классом, и можно работать с его объектами вместо объектов класса-родителя k, который теперь можно рассматривать как однородный. Данный подход обладает следующим недостатком: пусть k'' - неоднородный (возможно, финальный) класс, для которого k является предком (не обязательно непосредственным), тогда любой объект o'' класса k'' и любой объект o класса k находятся в естественном отношении порядка, однако объект o' класса k' уже находиться с объектом o'' в отношении порядка не будет (и, в частности, может обладать большим по включению набором прав доступа, чем o''), т.е. описанное нами сведение не является (в контекстном смысле) эквивалентным.

Для построения HRU-модели финально-неоднородной объектно-ориентированной системы с естественной иерархией определим следующие элементарные операции:

1. $Create(o^k, k)$ – создает объект o^k класса $k \in K$, при этом $O := O \cup \{o^k\}$, $o^k.M[] := k.M[]$, и в матрицу доступа каждого объекта $o'^{k'}$, $k \in K$ системы добавляется строка, соответствующая объекту o^k, причем $o'^{k'}.M[o^k, o'^{k'}.x] = o'^{k'}.M[k, o'^{k'}.x]$ для всех полей и открытых методов x объекта $o'^{k'}$.

2. $Destroy(o^k)$ – уничтожает объект o^k, при этом $O := O \setminus \{o^k\}$.

3.1 $Enter(r, o^k, o'^{k'}.f)$ – вносит право доступа r в $o'^{k'}.M[o^k, o'^{k'}.f]$, где o^k - объект класса k, $o'^{k'}$ - объект класса k'. При этом матрица доступа изменяется по правилу $o'^{k'}.M[o^k, o'^{k'}.f] := o'^{k'}.M[o^k, o'^{k'}.f] \cup \{r\}$. Данная элементарная операция может быть выполнена, только если k и k' - финальные классы, и выполнены следующие условия целостности:

а) $r \in k'.M[o^k, k'.f]$;

б) $r \in o'^{k'}.M[k, o'^{k'}.f]$.

3.2 $Enter(r, o^k, k'.f)$ – вносит право доступа r в $k'.M[o^k, k'.f]$, где o^k - объект класса k. При этом матрицы доступа изменяются по правилу: $k'.M[o^k, k'.f] := k'.M[o^k, k'.f] \cup \{r\}$, и $o'^{k'}.M[o^k, o'^{k'}.f] := o'^{k'}.M[o^k, o'^{k'}.f] \cup \{r\}$ для всех объектов $o'^{k'}$ класса k'. Данная элементарная операция может быть выполнена, только если k - финальный класс, а k' -однородный, и выполнены следующие условия целостности:

а) $r \in k'.M[k, k'.f]$;

б) для любого родителя k_2 класса k', обладающего полем f, $r \in k_2.M[o^k, k_2.f]$.

3.3 $Enter(r, k, o'^{k'}.f)$ – вносит право доступа r в $o'^{k'}.M[k, o'^{k'}.f]$, где $o'^{k'}$ – объект класса k'. При этом матрицы доступа изменяются по правилу $o'^{k'}.M[k, o'^{k'}.f] := o'^{k'}.M[k, o'^{k'}.f] \cup \{r\}$ и $o'^{k'}.M[o^k, o'^{k'}.f] := o'^{k'}.M[o^k, o'^{k'}.f] \cup \{r\}$

для всех объектов o^k класса k. Данная элементарная операция может быть выполнена, только если k' – финальный класс, а k – однородный, и выполнены следующие условия целостности:

а) для любого потомка k_1 класса k, $r \in o'^{k'}.M[k_1, o'^{k'}.f]$;

б) $r \in k'.M[k, k'.f]$.

3.4 $Enter(r, k, k'.f)$ – вносит право доступа r в $k'.M[k, k'.f]$, при этом матрицы доступа изменяются по правилу $k'.M[k, k'.f] := k'.M[k, k'.f] \cup \{r\}$ и $o'^{k'}.M[o^k, o'^{k'}.f] := o'^{k'}.M[o^k, o'^{k'}.f] \cup \{r\}$ для всех объектов $o'^{k'}$ класса k' и всех объектов o^k класса k. Данная элементарная операция может быть выполнена, только если k и k' – однородные классы и выполнены следующие условия целостности:

а) для любого потомка k_1 класса k, $r \in k'.M[k_1, k'.f]$;

б) для любого родителя k_2 класса k', обладающего полем f, $r \in k_2.M[k, k_2.f]$.

Операции *Delete*, *Grant* и *Deprive* для классов и объектов определяются аналогично операции *Enter* с учетом соответствующих условий целостности.

Определение 11. Под состоянием компьютерной системы в момент времени t будем понимать пару $Q(t) = (O(t), M(t))$, где $O(t) = \{o_j\}$ - множество всех объектов системы в момент времени t, а $M(t) = \{y.M[](t), y \in K \cup O\}$ - семейство всех матриц доступа классов и объектов системы в состоянии на момент t. Начальное состояние системы будем обозначать $Q(0)$.

Состояния компьютерной системы в модели HRU изменяются под воздействием запросов на модификацию матрицы доступа в виде команд $\gamma(x_1 : k_1, \ldots, x_m : k_m)$ следующего формата:

if – условная часть (представляет собой конъюнкцию значений логических выражений вида $r \in y'.M[y, y'.f]$ или $y'.M[y, y'.s] = 1$);

$then$ – последовательность элементарных операций.

Здесь $y, y' \in K \cup O\}$, а каждый аргумент x_i, $i = 1 \ldots m$ представляет собой либо объект, либо поле или функцию класса, либо поле или функцию объекта определенного класса $k_i \in K$, участвующую в качестве аргументов в условной части либо в элементарных операциях. Классы не являются аргументами команды, они определяют саму команду (как содержащиеся в теле команды операторы либо условия): так, например, если x_i - это объект, то он может быть только предзаданного класса k_i. Создаваемый операцией *Create* объект также не является аргументом команды, т.к. важен только его класс.

Напомним, что помимо уже входящих в команду условий каждая операция *Enter*, *Delete*, *Grant*, *Deprive* по умолчанию добавляет в if-часть команды свои условия целостности, соединенные операцией конъюнкции. Таким образом, если хотя бы одно условие целостности не будет соблюдено, вся команда не будет выполнена.

К тому же, не должно существовать команд, содержащих одновременно операции $Enter(r, y, y''.f)$ и $Delete(r, y, y'.f)$, либо $Grant(y, y''.s)$ и $Deprive(y, y'.s)$, либо $Enter(r, y', y.f)$ и $Delete(r, y'', y.f)$, либо $Grant(y', y.s)$ и $Deprive(y'', y.s)$, где y'' - потомок класса y', если $y'' \in K$, и y'' - представитель класса y', если $y'' \in O^{y''}$. Несоблюдение этого условия приведет к нарушению естественной

иерархии. Все прочие сочетания элементарных операций являются допустимыми, т.к. не могут привести к нарушению естественной иерархии.

Наиболее естественный способ избавиться от команд, способных нарушить естественную иерархию, – запретить таким операциям, как *Enter* и *Delete*, находиться в одной команде вообще. Однако такой шаг заметно уменьшит количество систем, попадающих под рассмотрение.

Определение 12. Будем говорить, что HRU-модель объектно-ориентированной системы с естественной иерархией почти монотонна, если ни одна из операций *Create*, *Enter*, *Grant* не может находиться в одной команде с одной из операций *Destroy*, *Delete*, *Deprive*.

Прежде чем приступить к обсуждению проблемы безопасности системы, перепишем пару определений для случая финально-неоднородных систем.

Определение 13. Будем говорить, что состояние финально-неоднородной системы $Q(t)$ допускает утечку права доступа $r \in R$, если существуют такие команда γ, класс либо объект y и поле $y'.f$ класса, либо объекта y', что $r \notin y'.M[y, y'.f](t)$, но $r \in y'.M[y, y'.f](t+1)$, где $Q(t) \to_\gamma Q(t+1)$.

Определение 14. Будем говорить, что состояние финально-неоднородной системы $Q(t)$ допускает утечку права вызова, если существуют такие команда γ, класс, либо объект y и метод $y'.s$ класса либо объекта y', что $y'.M[y, y'.s](t) = 0$, но $y'.M[y, y's](t+1) = 1$, где $Q(t) \to_\gamma Q(t+1)$.

Теорема 2. *Проблема r-безопасности системы является NP-разрешимой для почти монотонной финально-неоднородной объектно-ориентированной модели с естественной иерархией.*

Доказательство. Будем считать, что все объекты одного класса обладают полным доступом друг к другу. В противном случае наша задача не сильно усложнится.

Будем искать цепочку команд наименьшей длины, переводящую систему в состояние, допускающее утечку некоторого права r. По-прежнему считаем, что такая цепочка не должна содержать команд, не изменяющих ни одну матрицу доступа.

Допустим,

$$Q(0) \to_{\gamma_1} Q(1) \to \cdots \to_{\gamma_{T-1}} Q(T-1) \to_{\gamma_T} Q(T),$$

где $Q(T)$ допускает утечку права $r \in R$ в ячейку $M[](T)$.

Во-первых, в такой цепочке не может быть более одной «деструктивной» команды, содержащей операции *Destroy*, *Delete*, *Deprive* (одна такая команда может потребоваться в том случае, если изначально в матрице $M[](0)$ право r содержалось в ячейке, куда впоследствии произойдет его утечка). Также в ней содержится не более $|K|$ команд, содержащих только операции *Create* – по одному новому объекту o^k для каждого класса $k \in K$.

Под o^k будем понимать первый созданный объект класса k (возможно, он присутствовал в состоянии $Q(0)$ системы, но позже был удален единственной командой, содержащей операцию $Destroy$. Пусть теперь o'^k - новый объект класса k, созданный позже o^k. В аргументах всех последующих команд заменяем o'^k на o^k (сделать это возможно, т.к. они одного типа), при этом матрица доступов $o^k.M[](t)$ в любой момент времени будет содержать в точности те права доступа, которые содержали до замены $o^k.M[](t)$ и $o'^k.M[](t)$ в совокупности, поскольку в цепочке больше не встретятся «деструктивные» команды. Если утечка права связана с одним из новых объектов класса k, то она произойдет в новой цепочке не позже, чем в исходной цепочке, т.к. условием выполнения команды является только наличие права, а не его отсутствие. Конечно, при движении по цепочке новые объекты продолжат создаваться, но их можно игнорировать, поскольку в дальнейшем в цепочке никаких операций с ними не производится.

Таким образом, за время работы системы будут изменяться матрицы доступов не более чем $|O(0)| + |K|$ объектов, не более чем $|O(0)| + |K|$ строчек в каждой и не более, чем N столбцов, максимум R прав в каждой ячейке таблицы. Здесь $O(0)$ - множество объектов в состоянии $Q(0)$, а N - наибольшее количество полей и методов в классе. Поскольку каждая команда добавляет хотя бы одно право в какую-то ячейку какой-то матрицы (кроме единственной «деструктивной» команды), то, очевидно, не более чем за $2(|O(0)|+|K|)^2NR+1$ команд произойдет утечка права r. Итак, длина цепочки конечна.

Назовем две команды γ_1 и γ_2 условно-эквивалентными, если при замене в них всех аргументов каждого из классов k на объект o_k, определенный выше, команды будут совпадать по изменению матриц доступа во всех возможных состояниях Q системы. Условная эквивалентность является отношением эквивалентности и, таким образом, разбивает множество команд (вообще говоря, бесконечное) на классы эквивалентности. Таких классов эквивалентности – конечное число, т.к. каждый класс эквивалентности e может представлять команда γ_e, аргументами которой являются лишь поля или методы конечного числа классов объектно-ориентированной системы, а объекты, как однозначно определяемые своим классом, аргументами не являются. Количество классов же в объектно-ориентированной системе мы считаем величиной конечной и постоянной.

На самом деле команды γ_e не обязаны принадлежать системе. Но для цепочки команд представителем каждого класса эквивалентности можно всегда выбирать команду, наименьшую по максимальному числу объектов одного класса в строке аргументов. Таким образом, в допускающей утечку права r цепочке конечной длины на каждом месте может стоять команда из некоторого конечного множества. Получаем, что нам достаточно перебрать конечное множество цепочек.

Заметим, что если объекты одного класса обладают лишь частичным динамично меняющимся доступом к друг другу, в нашем доказательстве достаточно увеличить максимальное количество рассматриваемых новых объектов каждого класса до двух, т.к. возможна ситуация, в которой происходит утечка права доступа объекта класса k к полю другого объекта того же класса, и оба объекта

отсутствовали в $Q(0)$. Но это не повлияет на конечность задачи (придется лишь слегка изменить понятие условной эквивалентности).

■

Литература

1. Белим С.В., Белим С.Ю. Объектно-ориентированная модель компьютерной системы // Математические структуры и моделирование. 2008. Вып. 18. С. 98-101.
2. Белим С.В., Усов С.В. Объектно-ориентированный подход в построении политики безопасности // Математические структуры и моделирование. 2009. Вып. 20. С. 153-159.
3. Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. Communications of the ACM, 19(8):461–471, August 1976.
4. Гайдамакин Н.А. Разграничение доступа к информации в компьютерных системах. Уральский университет, 2003.

Математические структуры и моделирование
2010, вып. 21, с. 162–182

УДК 004.78:364.043.4

РАЗРАБОТКА САЙТА ПСИХОСОЦИАЛЬНОЙ АДАПТАЦИИ СТУДЕНТОВ ПЕРВОГО КУРСА ФКН ОМГУ ИМ. Ф.М. ДОСТОЕВСКОГО

Ю.П. Дубенский, С.Ю. Лаврова, Д.Н. Лавров

В данной работе рассматриваются задачи психосоциальной адаптации студентов первых курсов факультета компьютерных наук ОмГУ к учебному процессу посредством информационного сайта. Обосновывается актуальность создания такого сайта, разрабатывается структура, подбираются инструменты управления содержанием и само содержание. Устанавливаются роли пользователей сайта.

Введение

Развитие современного общества обострило проблемы социализации молодежи, вызвало необходимость искать оптимальные пути социальной адаптации студенчества. Разрешение данной проблемы, как части социальных преобразований, будет успешной, если человек будет рассматриваться, как часть социальной среды. Социальный организм, частью которого является молодой человек, непрестанно развивается. Что-то отмирает, что-то возникает заново, и только человек из своего понимания, из своего сознания и чувства может шаг за шагом направлять его развитие к идеалу – если, конечно, этот идеал у человека есть и достаточно отчетливо им осознается, если человек хочет этот идеал воплощать в жизнь.

Происходящие в обществе негативные изменения особенно болезненны для молодежи вследствие неустоявшегося мировоззрения, её динамично изменяющегося социального положения, возрастной восприимчивости окружающего мира, неустойчивости картины нравственно-эстетических ценностей.

Многочисленные примеры, факты из окружающей нас действительности часто свидетельствуют о растерянности молодых людей, об их отказе от поиска смысла жизни, идеала, пессимистическом восприятии окружающей среды. Это приводит к скептическому отношению к понятиям гражданского долга, социальной активности, ответственности.

Данное обстоятельство становится причиной падения духовности человека, снижения познавательного интереса, пренебрежительного отношения к нормам

Омский государственный университет им. Ф.М. Достоевского.
E-mail: socialrab@mail.ru, lavrov@omsu.ru

нравственности. Поэтому все большее распространение в молодежной среде получают порнография, спекуляция, пессимизм, социальная апатия, стремление жить только сегодняшним днем и т.д.

Сегодняшняя сфера образования, по словам известного отечественного педагога Б.Т. Лихачева, оказалась практически обесцеленной, идейно ослабленной, а педагоги – обезоруженными и обессиленными перед стихийными влияниями негативных явлений [2].

Система высшего образования не является исключением и не удовлетворяет в полной мере требованиям общества к социальной адаптации учащейся молодежи, сохранению и развитию нравственности и профессионализма будущих специалистов. Вуз сегодня слабо ориентирован на создание условий для оптимизации социальной адаптации молодежи в студенческой среде. А между тем рыночная экономика, в условиях которой живут и формируются, а впоследствии и будут работать в качестве специалистов нынешние студенты, выдвигает жесткое требование к специалисту как субъекту, владеющему не только адекватными времени знаниями, но и широкими возможностями выбора средств для решения возникающих проблем. Отсюда следует, что система высшего образования должна выступать гарантом педагогических условий социальной адаптации личности студента с учетом его интересов, способностей, запросов, ценностных ориентаций.

Однако на сегодняшний день именно этап адаптации студента к новым условиям его жизни является менее изученным в системе высшего образования. Известно, что обучение в вузе у молодого человека совпадает со стадией его развития, связанной с активным формированием его социальной зрелости, интенсивным нравственно-эстетическим становлением, моделированием профессионального жизненного пути, рефлексивным отношением к собственной жизнедеятельности. Понятно, что студент выступает не только объектом целенаправленного воздействия окружающей среды, но и активным субъектом формирования себя и окружающей среды.

Современный студенческий социум подвержен множественному влиянию извне, в том числе негативному. Поэтому не всегда внутренние и внешние взаимоотношения членов студенческого социума становятся гармоничными. Без разрешения данных проблем невозможно обеспечить оптимальность социальной адаптации студентов, на что и должна быть направлена деятельность разнообразных механизмов социализации будущих студентов.

При этом нужно отметить, что процесс социальной адаптации личности растущего человека, как правило, рассматривается в ходе изучения профессионально-педагогической деятельности учителей К.Д. Ушинским, Н.К. Крупской, П.П. Блонским, А.С. Макаренко, С.Т. Шацким, В.А. Сухомлинским.

Социально-психологический аспект адаптации учащихся исследуется в работах: С.Э. Артемова, Д.А. Андреевой, С.Д. Агарелян, Ю.С. Бабахай, Ф.Б. Березина, А.А. Гордон, О.И. Зотовой, И.К. Кряжевой и др.

Обосновывают необходимость целенаправленного педагогического управления процессом адаптации, предлагают пути и средства повышения ее эффек-

тивности И.И. Ляхов, Ф.С. Махов, Д.Д. Наурузбаев.

Проблема адаптации студентов к условиям обучения в высшей школе представляет собой одну из важных общетеоретических проблем, исследуемых в настоящее время.

Ученые (В.Г. Бочарова, Л.П. Буева, Б.З. Вульфов, М.П Гурьянова, В.И. Загвязинский, Г.В. Залевский, Н.М. Иовчук, Н.С. Морова, Л.Е. Никитина, Г.Н. Филонов, В.Т. Лисовский и др.) исследуют сущность социально-адаптационного процесса, обосновывают статус категории «социальная адаптация», изучают механизмы адаптации.

Актуальность проблемы определяется задачами оптимизации процесса адаптации студентов к процессу обучения в условиях современного вуза. В этой связи для ускорения процесса адаптации студентов к условиям обучения в высшей школе, к новому для них образу жизни и деятельности необходимо выявить психолого-педагогические условия оптимизации данного процесса, найти эффективные механизмы управления адаптационными процессами в вузе.

Одним из наиболее успешных средств адаптации студентов к процессу обучения в вузе является широкое использование и применение информационных технологий в учебном процессе.

Объектом исследования является процесс психосоциальной адаптации студентов к обучению в вузе, предметом —- социально-психологическая адаптация студентов посредством информационного сайта. Цель данной работы — разработка проекта Интернет сайта «Ступени» как средства психосоциальной адаптации студентов. Для достижения данной цели необходимо решить следующие задачи:

1. Проанализировать особенности психосоциальной адаптации студентов.
2. Провести анализ причин отчисления студентов.
3. Выявить уровень тревожности студентов-первокурсников.
4. Разработать проект Интернет-сайта социально-психологической адаптации студентов.

1. Анализ проблемы адаптации студентов первого курса

1.1. Понятие и основные характеристики процесса адаптации

В науке всегда существовала потребность в общепринятых понятиях. Исследования проблем адаптации в различных областях науки и техники показывают, что на роль такой общенаучной категории вполне обоснованно и логично может претендовать понятие «адаптация».

Прежде чем приступить к анализу различных подходов к адаптации, необходимо дать определение адаптации как социальному механизму. Существует множество определений адаптации как имеющих общий, очень широкий смысл, так и сводящих сущность адаптационного процесса к явлениям одного из множества уровней — от биохимического до социального.

Сам термин «адаптация» впервые появился в физиологии и использовался изначально в биологических науках. В научный оборот он был введен немецким физиологом Г. Аубертом и обозначал изменения (приспособительного характера) чувствительности кожных анализаторов к действию внешних раздражителей [16].

В литературе по данной тематике встречаются следующие определения адаптации:

К.К. Платонов считает, что адаптация — это приспособление органа, организма, личности или группы к изменению внешних условий. Он различает физиологическую, социально-психологическую и профессиональную адаптацию [3].

По мнению А.А. Налчаджяна, адаптация —- это тот социально-психологический процесс, который при благоприятном течении приводит личность к состоянию адаптированности [1].

В общей психологии А.В. Петровский, В.В. Богословский, Р.С. Немов практически одинаково определяют адаптацию как ограниченный, специфический процесс приспособления чувствительности анализаторов к действию раздражителя [10].

По определению В.И. Медведева «адаптация - это целенаправленная системная реакция организма, обеспечивающая возможность всех видов социальной деятельности и жизнедеятельности при воздействии факторов, интенсивность и экстенсивность которых ведет к нарушениям гомеостатического баланса». Эта реакция может иметь различную силу и интенсивность [6].

Д.П. Дербенев, на основе сравнительного анализа большого числа определений адаптации, предложил свое обобщенное понятие адаптации: «Адаптация есть особая форма отражения системами воздействия внешней и внутренней среды, заключающегося в тенденции установления с ним динамического равновесия» [7].

Психологические словари в общем случае определяют адаптацию как приспособление строения и функций организма, его органов и клеток к условиям среды [8–10, 12].

В более общих определениях понятия адаптации ему может придаваться несколько значений, в зависимости от рассматриваемого аспекта:

а) адаптация используется для обозначения процесса, при котором организм приспосабливается к среде;
б) адаптация используется для обозначения отношения равновесия (относительной гармонии), которое устанавливается между организмом и средой;
в) под адаптацией понимается результат приспособительного процесса;
г) адаптация связывается с какой-то определенной «целью», к которой «стремится» организм [5].

Анализ источников по исследуемой проблеме также свидетельствует о многообразии взглядов в определении понятия «социальная адаптация». Ключевым в определении содержания понятия «социальная адаптация» является процесс активного освоения личностью социальной среды, в котором личность высту-

пает как в качестве объекта, так и в качестве субъекта адаптации, а социальная среда является одновременно и адаптирующей, и адаптируемой стороной.

В Социологическом энциклопедическом словаре социальная адаптация рассматривается как процесс активного приспособления индивида или группы к определенным материальным условиям, нормам и ценностям социальной среды [9].

В Социологическом справочнике дается следующее определение понятия «социальная адаптация»: «Активное освоение личностью или группой новой для нее социальной среды» [12].

В Психологическом словаре под редакцией В.П. Зинченко, социальная адаптация рассматривается следующим образом: с одной стороны, как процесс активного приспособления индивида к условиям социальной среды, с другой - как результат этого процесса [8].

Согласно Д.В. Ольшанскому, социальная адаптация это вид взаимодействия личности или социальной группы с социальной средой, в ходе которого согласовываются требования и ожидания его участников. Важнейший компонент адаптации – согласование самооценок и притязаний субъекта с его возможностями и реальностью социальной среды, включающее также тенденции развития среды и субъекта [13].

По мнению Е.П. Никитина, Н.Е. Харламенковой, социальная адаптация — система методов и приемов, имеющих целью оказание социальной поддержки людям в процессе их социализации или приспособления к новым социальным условиям в связи с изменением социального статуса, жизненных утрат и неудач, а также неадаптированным личностям [14].

Рассматривая понятие «социальная адаптация», не следует отделять психологический аспект от социального, так как адаптация является комплексным феноменом — социальная среда, где вращается личность, подразделяется на предметную и личностную.

А.А. Налчаджян на основе анализа зарубежных и отечественных исследований трактует социально-психологическую адаптированность личности как особое состояние взаимодействия личности и группы, «когда личность без длительных внешних и внутренних конфликтов продуктивно выполняет свою ведущую деятельность, удовлетворяет в полной мере свои основные социальные потребности, в полной мере идет навстречу тем ролевым ожиданиям, которые предъявляет к ней эталонная группа, переживает состояние самоутверждения и свободного выражения своих творческих способностей» [1].

В нашей дальнейшей работе под словом адаптация мы будем понимать социально-педагогическую адаптацию — эмоционально-чувственное приспособление психики студентов при включении их в новые социальные условия.

Адаптация — одно из ключевых понятий в научном исследовании человеческой природы. Это естественный и необходимый компонент существования человека в системе «организм-среда», в системе «личность-социум», поскольку именно механизмы адаптации, имеющие эволюционные корни, обеспечивают возможность выживания человека.

Одним из вариантов адаптации человека в обществе является его профес-

сионализм, который выступает в качестве ценного ресурса для выживания и эффективной жизнедеятельности. Стремление к овладению профессией является одним из побуждающих факторов поступления выпускников школ в высшие учебные заведения. Поступление в вуз сопровождается переходом в новую систему образования, новую социальную среду, что является делом сложным и подчас болезненным, вызывающим необходимость адаптации первокурсников к учебному процессу.

Новая обстановка, новый режим, иные учебные нагрузки и требования, новые отношения, новая социальная роль, новый уровень отношений с родителями, иное отношение к себе — это далеко не полный перечень изменений, обретающих особую остроту в первый год обучения. У первокурсников происходит смена привычного образа жизни, что автоматически включает адаптационный процесс.

Адаптация студентов к обучению в вузе представляет собой многоуровневый процесс, который включает составные элементы социально-психологической адаптации и способствует развитию интеллектуальных и личностных возможностей студентов.

В свою очередь, адаптационный процесс связан с решением целого спектра различных проблем. Одной из центральных социально-психологических проблем процесса адаптации является освоение новой социальной роли — роли студента. У бывшего школьника навыков выполнения такой роли нет. И отсюда целый комплекс как внутренних, так и внешних конфликтов, связанных с трудностями принятия и дальнейшего выполнения норм, соответствующих социальной роли студента. Студенты-первокурсники методом проб и ошибок пытаются освоить ожидаемое от них поведение и на его основе строить дальнейшие взаимоотношения со сверстниками и преподавателями.

1.2. Особенности адаптации студентов в условиях вуза

1.2.1. Психологические особенности студенческого возраста

Решая конкретные вопросы по повышению эффективности подготовки специалистов, снижению социальной дезадаптации в процессе вузовского обучения, необходимо помнить о сущности профессионального становления специалистов, о возрастных особенностях изучаемого контингента, о психологическом аспекте учебной деятельности в вузе, специфике формирования специалиста.

Как отмечает И.С. Кон, ведущей сферой деятельности в студенческом возрасте становится труд с вытекающей отсюда дифференциацией профессиональных ролей. Об этой возрастной группе нельзя говорить «вообще»: ее социально-психологические свойства зависят не столько от возраста, сколько от социально-профессионального положения. Образование, которое продолжается и на этом этапе развития, является уже не общим, а специальным, профессиональным, причем сама учеба в вузе может рассматриваться как вид трудовой деятельности [18].

По словам Э. Шпрангера [19], данный период характеризуется тем, что на первый план выступают «кризис оторванности», чувство одиночества, которые

во многом определяют особенности эмоциональных состояний юношей: склонность к крайним эмоциональным переживаниям, негативный эмоциональный фон, подавленность.

Существует и другая точка зрения: по словам А. Гезелла [12], в юношеском возрасте наступает равновесие: мятежность уступает место жизнерадостности, значительно увеличивается внутренняя самостоятельность, эмоциональная уравновешенность, общительность, устремленность в будущее.

Характерной чертой развития в этом возрасте является усиление сознательных мотивов поведения. Заметно укрепляются такие качества, как целеустремленность, решительность, настойчивость, самостоятельность, инициатива, умение владеть собой, стремление к самопознанию и самоопределению в качестве субъекта социальной жизни, а также активное взаимодействие с окружающим миром. Мировоззренческое самоопределение включает в себя социальную ориентацию личности, формирование жизненных планов, формирование собственной системы ценностей и собственный интеллектуальный поиск. Самоопределение является само по себе очень сложным процессом, который сопровождается перестройкой внутренней организации личности и предъявляет к юношам особые требования. Таким образом, процесс взросления сопровождается значительными психологическими трудностями, которые ещё в большей степени обостряют проблему адаптации первокурсников к учебе.

Социально-психологическая адаптация предполагает осознание социального статуса, ролевого поведения в системе межличностных отношений, формирования и реализации определенных ценностных ориентаций, групповых норм и т.д. Данные вопросы играют особую роль при подготовке специалиста и его социальной адаптации.

1.2.2. Особенности адаптации студентов к обучению в вузе

Адаптация студентов к обучению в вузе имеет свои особенности. Вхождение молодых людей в систему вузовского обучения, приобретение ими нового социального статуса студента требуют от них выработки новых способов поведения, позволяющих им в наибольшей степени соответствовать своему новому статусу. Такой процесс приспособления может проходить достаточно длительное время, что может вызвать у человека перенапряжение как на психологическом, так и на физиологическом уровнях, вследствие чего у студента снижается активность, и он не может не только выработать новые способы поведения, но и выполнить привычные для него виды деятельности.

Специфика процесса адаптации в вузах определяется различием в методах обучения и в его организации в высшей школе, что порождает своеобразный отрицательный эффект — дидактический барьер. Первокурсникам недостает различных навыков и умений, которые необходимы в вузе для успешного овладения программой. Приспособление к новым условиям требует много сил, из-за чего возникают существенные различия в деятельности и результатах обучения в школе и вузе.

Известно, что процесс адаптации к обучению в вузе - явление весьма слож-

ное, многогранное, завершающееся к концу третьего курса. Выделяют две стороны социальной адаптации студентов:

- профессиональную адаптацию, под которой понимается приспособление к характеру, содержанию, условиям и организации учебного процесса, выработка навыков самостоятельности в учебной и научной работе, формирование нового отношения к профессии;
- социально-психологическую адаптацию – приспособление индивида к группе, взаимоотношениям в ней, приспособление к новым условиям быта в студенческих общежитиях, новым образцам студенческой культуры, новым формам использования свободного времени, выработка собственного стиля поведения.

Другими словами, на протяжении начальных курсов складывается студенческий коллектив, формируются навыки и умения рациональной организации умственной деятельности, осознается призвание к избранной профессии, вырабатывается оптимальный режим труда, досуга и быта, устанавливается система работы по самообразованию и формированию профессионально значимых качеств личности.

Социально-профессиональную адаптацию принято рассматривать в трех аспектах:

1) медико-биологический аспект (результат приспособления организма индивида и его психики к условиям профессиональной деятельности и общения);
2) психолого-педагогический аспект (процесс социализации индивида в труде и результат этого процесса);
3) социологический аспект (особый этап жизненного пути молодого человека, совпадающий по времени с окончанием обучения в системе народного образования и началом профессиональной трудовой деятельности) [17].

Профессиональная адаптация — предпосылка активной деятельности и необходимое условие ее эффективности. Она предполагает приспособление к характеру, содержанию, условиям и организации учебного процесса, выработка навыков самостоятельности в учебной и научной работе. Социально-психологическая адаптация – приспособление индивида к группе, взаимоотношениям с ней, выработка собственного стиля поведения.

В этом заключено положительное значение адаптации для успешного функционирования индивида (студента) в той или иной социальной роли.

1.3. Основные формы адаптации студентов

Различают три формы адаптации студентов-первокурсников к условиям вуза:

- адаптация формальная, касающаяся познавательно-информационного приспособления студентов к новому окружению, к структуре высшей школы, к содержанию обучения в ней, ее требованиям к своим обязанностям;
- адаптация общественная, то есть процесс внутренней интеграции (объединения) групп студентов-первокурсников и интеграция этих же групп со студенческим окружением в целом;

- адаптация дидактическая, касающаяся подготовки студентов к новым формам и методам учебной работы в высшей школе.

Для успешной адаптации необходимым является проявление активной позиции, которая должна быть не только у преподавателя, но и у студента. Студент должен сам находить и выбирать для себя способы и пути достижения той или иной образовательной цели, а преподаватель – создавать для этого условия.

Именно на первом курсе формируется отношение молодого человека к учебе, к будущей профессиональной деятельности, продолжается активный поиск себя. Даже отлично окончившие среднюю школу, на первом курсе не сразу обретают уверенность в своих силах. Первая неудача порой приводит к разочарованию, утрате перспективы, отчуждению, пассивности.

В настоящее время проблема адаптации студентов на начальном этапе профессиональной подготовки занимает одно из значимых мест в педагогической науке и практике. Это обусловлено тем, что система образования высшей школы оказалась не подготовленной к трансформациям общественно-политической жизни, когда одни только знания в традиционном понимании не могут выступать в качестве средства успешной адаптации вчерашних школьников.

1.4. Типичные трудности адаптации студентов в условиях вуза

Хотелось бы подробнее остановиться на социальной и учебной адаптации, связанной именно с обучением в вузе. Постараюсь перечислить трудности, с которыми приходится чаще всего сталкиваться студентам младших курсов в своей учебе:

- неумение ориентироваться в расписании, аудиториях, корпусах, электронном каталоге библиотеки;
- неумение правильно планировать свое время;
- отсутствие навыков работы с первоисточниками;
- плохая подготовка, полученная в школе;
- несамостоятельность;
- неуверенность в себе;
- беспокойство и страх перед сдачей сессии.

Из педагогических проблем следует отметить принципиальное отличие учебных нагрузок и форм организации учебной деятельности в вузе от таковых в школе. Все это вызывает дополнительное напряжение и повышает тревожность у первокурсников, усугубляя проблему адаптации.

Кроме того, в ходе первого года обучения могут возникнуть профессиональные сложности, связанные с осознанностью выбора будущей профессии. Вполне распространенное явление, когда через некоторое время после поступления в вуз студент понимает, что сделал неверный выбор. Совершенно очевидно, что такой поворот событий не способствует успешной адаптации студентов к высшей школе и учебному процессу.

Необходимо указать также на экономические факторы, влияющие на адаптацию первокурсников. В условиях перехода к рыночной экономике наблюдается тенденция ухудшения экономического положения студентов вузов. По этой

причине многие студенты с первых курсов вынуждены зарабатывать на жизнь, что в свою очередь, ставит перед ними ещё более трудные задачи и усложняет процесс адаптации. Часть студентов идет зарабатывать деньги, ещё не адаптировавшись к новым условиям и нагрузкам. Отсюда пропуски занятий, плохая учеба, несданная сессия и исключение из вуза как показатели дезадаптированности студента.

Значительное влияние на успешность процесса адаптации первокурсников оказали бы помощь и поддержка со стороны взрослых людей (психологов, кураторов), создающих условия, при которых происходила бы не только адаптация студентов к новому для них образовательному процессу, но и формировалась их готовность максимально раскрыть свои способности через осмысление собственного потенциала и возможностей его реализации.

Во многих вузах существует институт кураторства. Часто он воспринимается либо как формализованная структура, либо, особенно в понимании студентов, как «нянька», готовая решать любые возникающие проблемы. Задачи психолого-образовательного сопровождения каждый из кураторов определяет для себя сам или, более того, не определяет их вовсе, смутно догадываясь, какова его роль в работе со студентами. В нашем вузе такой структуры нет совсем. Хотя, на наш взгляд, институт кураторства нам просто необходим.

Хотелось обозначить цель кураторской деятельности как создание условий для учебной и социальной адаптации студентов младших курсов. Степень участия куратора в процессе адаптации студентов различна. Это зависит от уровня подготовленности выпускника средней школы, от его личностных характеристик, от сформированности мотивационной сферы, от его ожиданий и т.д. Кто-то нуждается в каждодневной помощи и контроле, а кому-то достаточно лишь обозначить уровень требований, перспектив. Следовательно, куратору необходимо владеть приемами диагностики, прогнозирования, профилактики, что возможно только при тесном взаимодействии с психологической службой вуза, готовой консультировать кураторов по различным проблемам адаптации, обучения и профессионального становления студентов, проводить диагностические исследования проблем, совместно организовывать мероприятия, направленные на профилактику учебной и социальной дезадаптации.

Из вышеназванных проблем и особенностей динамического процесса адаптации становится очевидным, что далеко не все студенты, попадая в атмосферу вуза, способны и могут быстро адаптироваться. Так, наблюдения Н. Ханчук показывают, что на втором курсе каждый четвертый студент не адаптирован к вузовской среде. Косвенное свидетельство этого -– высокий процент студентов, находящихся в академических отпусках, обучающихся повторно. Прямое свидетельство неполной адаптации студентов к учебе в вузе -– отсутствие устойчивых навыков планомерной, систематической учебы [14]. В результате затянувшейся адаптации может также происходить снижение успеваемости студентов, возникновение различных личностных проблем, ухудшение здоровья.

На наш взгляд, первый год обучения в большей степени решает задачу закладки фундамента для профессиональной подготовки в последующие годы студенческой жизни. Таким образом, успешное прохождение этого этапа яв-

ляется важной предпосылкой для дальнейших достижений студента. Отсюда возникает необходимость комплекса мероприятий оптимизации адаптационного процесса именно на первом курсе, что поможет студентам быстрее пройти этот нелегкий период.

1.5. Мероприятия, проводимые в системе вуза и, в частности, факультета компьютерных наук для успешной адаптации первокурсников

В условиях становления в России гражданского общества и правового государства особая роль отводится воспитательной работе со студентами.

Воспитание в вузе — органически связанная с обучением целенаправленная и систематическая деятельность, ориентированная как на формирование социально значимых качеств, установок и ценностных ориентаций личности, так и на создание благоприятных условий для всестороннего гармонического, духовного, интеллектуального и физического развития, самосовершенствования и творческой самореализации личности будущего специалиста.

Цель воспитательной работы — формирование гармонично развитой, творческой и высоконравственной личности будущего специалиста, способного успешно действовать в условиях конкурентной среды, обладающего высокой культурой и гражданской ответственностью за принимаемые решения, обладающего такими личностными качествами, как:

- нравственность,
- интеллигентность,
- патриотизм,
- стремление к здоровому образу жизни,
- профессиональная компетентность,
- социальная активность,
- предприимчивость,
- гражданская зрелость,
- способность к сотрудничеству и межкультурному взаимодействию.

Задачи воспитательной работы.

1. Создание условий для активной жизнедеятельности студентов, для гражданского самоопределения и самореализации, для максимального удовлетворения потребностей студентов в интеллектуальном, культурном, духовно-нравственном и физическом развитии, в связи с чем требуется:
   - создание полноценной социально-педагогической воспитывающей среды;
   - создание в университете социально-психологического климата, атмосферы творчества, демократии и гуманизма;
   - сохранение и приумножение историко-культурных и научных традиций университета, преемственности, формирование чувства университетской корпоративности и солидарности.
2. Формирование у студентов гражданской позиции и патриотического сознания, правовой и политической культуры.

3. Приобщение студентов к общечеловеческим ценностям и высоким гуманистическим идеалам, воспитание нравственности и интеллигентности.
4. Воспитание у студентов потребности к труду, к здоровому образу жизни, нетерпимого отношения к наркотикам, пьянству, антиобщественному поведению.
5. Воспитание у студентов потребности к саморазвитию, формирование умений и навыков управления коллективом в различных формах студенческого самоуправления.
6. Обеспечение социально-психологической поддержки студентов, адаптация первокурсников и иногородних студентов к изменившимся условиям жизнедеятельности с целью вхождения в университетскую среду.

Важной традицией студенчества является организация внеучебных мероприятий, направленных на выполнение одной из главных задач учебных заведений — гражданско-патриотическое и трудовое воспитание молодежи путем привлечения к активному участию в общественной жизни.

Направления воспитательной деятельности.

1. Проведение культурно-массовых, спортивно-оздоровительных, информационно-просветительных мероприятий, организация досуга студентов.
2. Создание и организация работы творческих, спортивных, интеллектуальных объединений студентов и преподавателей.
3. Организация гражданского и патриотического воспитания студентов.
4. Организация работы по профилактике правонарушений, наркомании и ВИЧ-инфекции среди студентов.
5. Организация психологической поддержки студентов, консультационной помощи.
6. Воспитательная работа в сфере профессиональной подготовки студента, работа по обеспечению вторичной занятости студентов.
7. Пропаганда физической культуры, формирование здорового образа жизни.
8. Содействие в работе студенческих общественных организаций, клубов и объединений.
9. Информационное обеспечение студентов, поддержка и развитие студенческих средств массовой информации.
10. Поиск и внедрение новых технологий, форм и методов внеучебной деятельности, наиболее эффективных путей и средств воспитания студентов.
11. Создание систем морального и материального стимулирования преподавателей и студентов, активно участвующих в организации внеучебной работы.
12. Развитие материально-технической базы и объектов, предназначенных для организации внеучебных мероприятий.

Каждый год в университете проводятся различные мероприятия по адаптации студентов. Отмечается также день российского студента (ректорский бал). Организатором выступает отдел по внеучебной работе ОмГУ. На факультете

компьютерных наук также существуют несколько видов внеучебной деятельности.

1. Общественно-организационная деятельность (помощь в организации и проведении семинаров, «круглых столов», конференций на факультете, олимпиад по программированию; помощь в проведении традиционных мероприятий факультета: выезд осенью с 1 курсом на природу, клятва студентов ФКН и др.)
2. Общественно-спортивная деятельность (участие во внутривузовских, районных, городских спортивных мероприятиях и др.)
3. Общественно-благотворительная деятельность (самостоятельное проведение благотворительных акций, рок-концертов в память жертвам СПИДа др.)
4. Общественно-творческая деятельность (организация и проведение тематических мероприятий на факультете «Посвящение в студенты», «День открытых дверей», организация профориентационной выставки «Тебе молодой»; организация внутригруппового досуга и др.)

Осуществление работы в данных направлениях позволяет студентам:

– раскрыть свои внутренние личностные ресурсы;
– обогатить опыт профессионального взаимодействия с социумом на микро-, макроуровне и выстроить свою профессиональную перспективу;
– занимать активную позицию в жизни факультета, университета.

Культурно-досуговая деятельность осуществляется через проведение следующих мероприятий:

– межгрупповые встречи со студентами старших и параллельных курсов факультета;
– летопись курса (составление фотоальбома; компьютерной и стендовой презентаций о жизни факультета; создание видео о жизни факультета и др.).

Осуществление данных мероприятий при работе со студентами позволяет:

1) сформировать атмосферу доверия и взаимопомощи;
2) создавать ситуации сотрудничества;
3) способствовать сплочению студентов;
4) содействовать релаксации студента внутри группы, курса, факультета;
5) расширять и углублять межличностные отношения студентов.

Существует ряд мероприятий, проведение которых является неотъемлемой частью функционирования студенческих групп. К ним относят:

- вечера, посвященные знаменательным датам (Новый год, Осенний бал);
- смотры-конкурсы, творческие конкурсы: «КВН», «ЧТО? ГДЕ? КОГДА?»;
- спортивные соревнования по волейболу, футболу, баскетболу, шахматные турниры между группами, курсами, деканатом и студентами, факультетами.

Проведение таких мероприятий является залогом создания благополучного микроклимата (курса, факультета), тем самым способствуя сплочению студенческого коллектива и организации студенческого самоуправления.

Анализ специальной литературы по проблеме психосоциальной адаптации

студентов 1 курса к обучению в вузе, имеющих возрастные границы с 17-18 лет, показал, что бывшие школьники, попав в стены высшего учебного заведения, сталкиваются с множеством психологических, социальных и экономических проблем.

Адаптация студентов к обучению в вузе представляет собой многоуровневый процесс, который включает элементы социально-психологической адаптации и способствует развитию интеллектуальных и личностных возможностей студентов.

2. Разработка интернет-сайта по психосоциальной адаптации студентов «Ступени»

2.0.1. Цели и задачи создания сайта

Цели проекта:

- создание и развитие нового вида социальных услуг для студентов, имеющих личностные проблемы в адаптации к условиям вуза;
- оказание психологической поддержки в преодолении трудностей личного роста, во время учебы и сдачи сессий, коррекции неадекватных способов реагирования в жизненных и учебных проблемных ситуациях посредством практической консультативной помощи в Интернет-пространстве, способствующей саморазвитию студента.

Задачи:

- разработать и внедрить проект Интернет-сайта психосоциальной адаптации студентов 1 курса ФКН «Ступени»;
- обучить студентов приемам снятия психологического напряжения в период экзаменационной сессии, методам саморегуляции.

Целевая группа: студенты первого курса высшего профессионального образовательного учреждения в возрасте 17-18 лет.

Контроль за результативностью проекта: Проект будет оцениваться педагогическим составом, включающим: зам. декана по учебной работе, зам. декана по воспитательной работе, члена учебно-методической комиссии и студентов профсоюзного комитета ФКН на основе организации экспертной оценки.

2.1. Анализ причин отчисления студентов

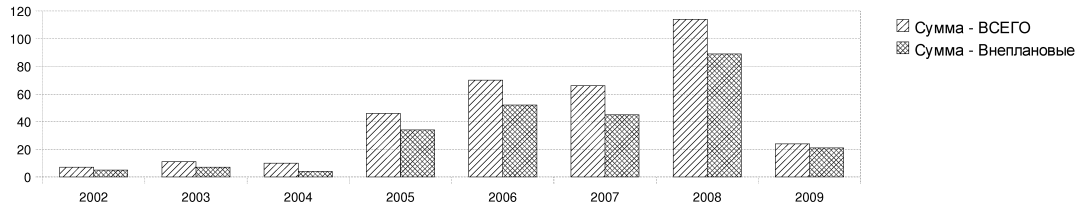
Для анализа причин отчисляемости студентов ФКН ОмГУ им. Ф.М. Достоевского были использованы данные корпоративной информационной системы «Студент», разработанной информационно-вычислительным центром ОмГУ.

Исходя из полученных данных, была построена сводная таблица №1, из которой видно, что основной причиной является отчисление студентов за академическую неуспеваемость. Цифры отчисленных за академическую неуспеваемость превышают в четыре раза другие виды отчислений.

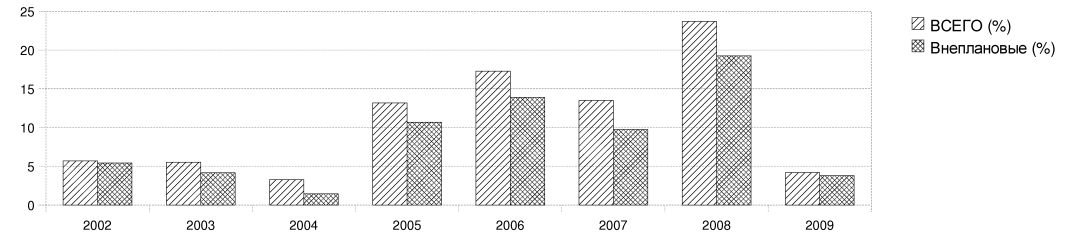
Если рассмотрим процент отчисляемых студентов на ФКН по годам (рис. 1), то становится очевидным, что количество и отчисленных студентов за академическую неуспеваемость увеличивается. Это связано с тем, что это поколение

90-х, которое начало поступать в 2006 г. В 90-х годах начался процесс гуманитаризации образования. Были сокращены часы в школьных программах по математике, физике и другим естественнонаучным дисциплинам. Общая неблагоприятная обстановка в стране сказалась на психологическом состоянии детей. Начался демографический спад. Видимое уменьшение отчисляемости студентов в 2009 объясняется доступностью данных только за зимнюю сессию.

Многие дети, обучаясь в школе, привыкли к постоянному контролю со стороны учителей и родителей. В высшем учебном заведении контроль ведется только в период сессий. Студентам часто не объясняют их обязанности и права, не разъясняют положение о текущей аттестации и соответствующих пунктов Устава университета. Все это способствует тому, что неустоявшаяся психика первокурсников подвергается соблазну не посещать занятия. Поэтому необходимо проводить всеми доступными средствами информационную разъяснительную работу по учебному процессу. В том числе и на сайте.



А. Общее количество отчисленных студентов за академическую неуспеваемость на ФКН (чел.)



Б. Общее количество отчисленных студентов за академическую неуспеваемость на ФКН (%)

Рис. 1. Отчисляемость студентов 1 курса ФКН ОмГУ

2.2. Структура сайта «Ступени». Описание функционалов и типовых страниц

Структура сайта разработана на основе интеллект-карты (см. рис. 2).

Диаграмма связей, известная также как интеллект-карта (англ. Mind map), — способ изображения процесса общего системного мышления с помощью схем. Также может рассматриваться как удобная техника альтернативной записи иерархических списков. Диаграмма связей реализуется в виде древовидной схемы, на которой изображены слова, идеи, задачи или другие понятия, связанные

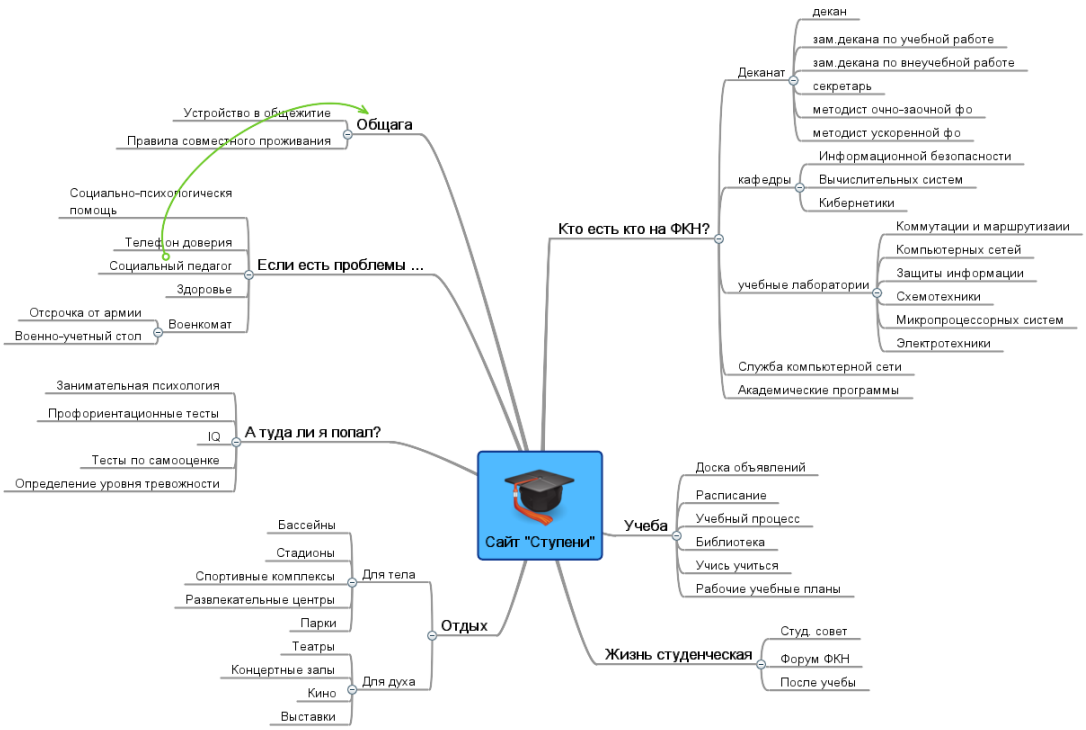


Рис. 2. Интеллект-карта структуры сайта «Ступени»

ветвями, отходящими от центрального понятия или идеи. В основе этой техники лежит принцип «радиантного мышления», относящийся к ассоциативным мыслительным процессам, отправной точкой или точкой приложения которых является центральный объект. Радиант — точка небесной сферы, из которой как бы исходят видимые пути тел с одинаково направленными скоростями, например метеоров одного потока. Это показывает бесконечное разнообразие возможных ассоциаций и, следовательно, неисчерпаемость возможностей мозга. Подобный способ записи позволяет диаграмме связей неограниченно расти и дополняться. Диаграммы связей используются для создания, визуализации, структуризации и классификации идей, а также как средство для обучения, организации, решения задач, принятия решений, при написании статей. Иногда в русских переводах термин может переводиться как «карты ума», «карты разума», «интеллект-карты», «карты памяти», или «ментальные карты». Наиболее адекватный перевод — «схемы мышления» [20].

Структура сайта подразумевает наличие семи основных разделов, которые направлены на решение следующих задач:

- познакомить со структурой факультета;
- выработать умение ориентироваться в учебном процессе;
- стимулировать активное участие в учебной и внеучебной деятельности;
- пропагандировать здоровый образ жизни;
- знать куда обращаться за помощью в трудных жизненных ситуациях;
- познакомить с правилами совместного проживания в общежитии (для

иногородних студентов).

Выделены следующие разделы структуры сайта:

1. Что такое ФКН?.
2. Кто есть кто на ФКН.
3. Учеба.
4. Жизнь студенческая.
5. А туда ли я попал?.
6. Если есть проблемы.
7. Отдых.
8. Общага.
9. ЧаВО.

В соответствии с задачами разделы сайта состоят из следующих блоков:

1. Что такое ФКН? Содержит две рубрики.
   - А. Университет: почему имени Ф.М. Достоевского; ОмГУ в цифрах; администрация ОмГУ; учебные подразделения ОмГУ; Карта корпусов.
   - Б. Деканат: декан, заместители декана, секретари); академические программы.
2. Кто есть кто на ФКН? Раздел включает в себя организационную структуру факультета: деканат, кафедры, учебные лаборатории, службу компьютерной сети, Сетевую академию Cisco.
3. Учеба. В этом разделе можно найти: расписание занятий, доску объявлений, описание учебного процесса (выдержки из положения о текущей аттестации), информацию о библиотеках, подраздел деканат отвечает на вопросы, информация для дипломников и курсовиков, рабочие учебные планы.
4. Жизнь студенческая. Раздел описывает: чем занимаются студенты на ФКН в свободное от учебы время, способы общения в сети, состав студенческого совета.
5. А туда ли я попал? Блок занимательной психологии: профориентационные тесты; тесты, направленные на определение уровня тревожности, степени социальной адаптации, уровня самооценки, уровня общительности. Задача этого блока заключается в развитии самопознания личности.
6. Если есть проблемы. Блок полезной информации, содержащий данные о всевозможных видах предоставляемой социально-психологической помощи на базе социальных центров города Омска. Также в данном блоке указан номер телефона доверия.
7. Отдых. В этом блоке содержится информация о культурно-досуговых местах города Омска.
8. Общага. Содержит общие положения, порядок предоставления помещений и заселения в студенческое общежитие, порядок прохода в общежитие, права и обязанности проживающих в студенческом общежитии, ответственность за нарушение настоящих Правил, порядок выселения проживающих из студенческого общежития.
9. ЧаВО. Название этого раздела — аббревиатура словосочетания «**Ча**сто

задаваемые **В**опросы и **О**ответы». Содержит полезную информацию о том, как оформить академический отпуск, как продлить сроки сдачи сессии, как перевестись на бюджет. Описывает процедуру отчисления, перевод на другую форму обучения. Содержит полезные телефоны.

2.3. Стилевые решения дизайна. Обоснование названия. Обоснование выбора цвета

Структура сайта должна быть в первую очередь удобна. Психологи утверждают, что человек, впервые попавший на заглавную страницу сайта, принимает решение j дальнейшем просмотре сайта около 8 секунд. Основная функция сайта – удовлетворять информационным запросам потенциального клиента или развлекать его, поэтому если пришедший по ссылке посетитель не находит ничего необходимого или интересного за первые 8 секунд, то он покидает этот сайт навсегда. Все разделы сайта и информация должны быть легко доступны, а навигация эргономична. Необходимо позволить зашедшему самостоятельно регулировать глубину просмотра.

Кроме того, очень важно, чтобы сайт имел название, ёмко отражающего его содержание и концепцию. Мы остановили свой выбор на названии «Ступени». Это название выражает то, что адаптированный к учебному процессу студент после успешной сдачи сессий будет продвигаться вперед, переходить с курса на курс, словно по ступеням лестницы, ведущей вверх, к достижению успеха в будущей профессии, карьерного роста.

При изготовлении сайта важно учитывать не только его тематику, но и эстетические предпочтения основной возрастной аудитории.

Чем моложе предполагаемая целевая аудитория сайта, тем выше вероятность того, что ей понравится яркий и сочный дизайн сайта, наполненный различными спецэффектами, забавными рисунками и иными новшествами.

Цвет также влияет на настроение и самочувствие. Поэтому при создании сайта очень важно учитывать:

- особенности фирменного стиля, на основании которого разрабатывается сайт;
- физиологические и психологические особенности восприятия цветовой гаммы людьми.

Для сайта были выбраны зеленый и оранжевые цвета, потому что зеленый — цвет природы, естества, самой жизни, весны. Тот, кто его предпочитает, боится чужого влияния, ищет способ самоутверждения, так как для него это жизненно важно. Тот, кто его не любит, страшится житейских проблем, превратностей судьбы, вообще всех трудностей. Зеленый цвет содержит в себе скрытую потенциальную энергию, отражает степень волевого напряжения. Его предпочитают способные, уверенные в себе, стремящиеся к самоутверждению и уравновешенные люди. Природа, здоровая среда, восстановление, молодость, бодрость, весна, щедрость, плодородие, ревность, неопытность, зависть. Оранжевый — любимый цвет людей, обладающих интуицией и страстных мечтателей. Оранжевый цвет - самый динамичный, молодежный и веселый цвет. Стимулирует чувства и

ускоряет сердцебиение, обостряет восприятие и способствует разрешению сложных ситуаций, задач и проблем. Жизнерадостный и импровизированный. Цвет создает чувство благополучия и счастья. Оказывает благоприятное воздействие на работоспособность. Он означает энергию, баланс, тепло, энтузиазм, оживленность. Считаем, что именно эти цвета будут соответствовать студенческому духу.

Актуальность проблемы проекта обоснована существующими психологическими, социальными и экономическими трудностями, с которыми сталкиваются студенты в повседневности, и отсутствием грамотной всесторонней психологической поддержки, позволяющей наиболее мягко преодолеть трудности, возникающие в процессе учебной деятельности.

Актуальность проекта интернет-сайта оценивалась педагогическим составом, включающим заместителя декана по учебной работе, заместителя декана по воспитательной работе, доцента кафедры информационной безопасности, на основе организации экспертной оценки по следующим критериям: актуальность проекта; оригинальность предложенного решения; соответствие форм и методов реализации проекта заявленной тематике; соответствие оформления проекта международным стандартам; правильность и полнота использования ресурсов.

Исходя из анализа проведенной экспертной оценки интернет-сайта «Ступени», можно утверждать, что данный проект является актуальным и готов для внедрения в учебный процесс ФКН ОмГУ им. Ф.М. Достоевского.

Заключение

В ходе проделанной работы были получены следующие результаты:

1. Проведенный анализ отчисляемости студентов на ФКН ОмГУ показывает рост отчисляемых студентов за академическую неуспеваемость в процентном отношении от общего числа обучаемых, из чего следует, что необходима организация мер по улучшению и оптимизации психосоциальной адаптации студентов.
2. Первым шагом в этом направлении является создание информационного центра, каким является веб-сайт, на котором размещены вспомогательные материалы, связанные с адаптацией.
3. Разработана структура сайта.
4. Сформулирована и обоснована основная концепция сайта.
5. Обоснован выбор элементов дизайна с психологической точки зрения (выбор основных цветов, оформление страниц).
6. Реализован и внедрен веб-сайт «Ступени» на ФКН на основе системы управления содержанием Joomla 1.5.
7. Предполагается, что контроль и мониторинг за выполнением программы адаптации будет производиться посредством анкетирования, проводимого в бумажном виде 2 раза в год, а так же непосредственно на самом сайте.
8. На момент написания статьи решаются технические проблемы размещения сайта под доменным именем `http://www.fkn-omsk.ru`.

Литература

1. Налчаджян А.А. Социально-психическая адаптация личности (формы, механизмы и стратегии). Ереван: Издательство АН Армянской ССР, 1988.
2. Лебедева Е.А. Педагогические условия социальной адаптации студентов технического вуза : Дис. канд. пед. наук : 13.00.01 : Калуга, 2001 204 с. РГБ ОД, 61:01-13/1802-9
3. Платонов К.К. Словарь системы психологических понятий. 2-е изд. М.: Высшая школа, 1984.
4. Российский статистический ежегодник 2003: Статистический сборник / Госкомстат России. М.: Госкомстат РФ, 2003.
5. Немов Р.С. Психология: учеб. для студентов в высш. пед. учеб. завед. / Р.С. Немов. М.: 1994. 576 с.
6. Медведев В.И. О проблеме адаптации // Компоненты адаптационного процесса Л., 1984.
7. Дербенев Д.П. Социальная адаптация подростков// Социологический журнал. 1997. № 1/2.
8. Психологический словарь / Под ред. В.П. Зинченко. – М.: Педагогика-пресс, 1996.
9. Социологический энциклопедический словарь / Ред.-координатор Г.В. Осипов. М., 1998.
10. Психологический словарь / Общ. ред. А. В. Петровского., М. Г. Ярошевского. М., 1990г.
11. Мудрик А.В. Социальная педагогика. М., 2002.
12. Социологический справочник / Под ред. В.И. Воловича. Киев: Изд-во политической литературы Украины, 1990.
13. Ольшанский Д.В. Адаптация социальная // Философский энциклопедический словарь / Ред. кол.: С.С. Аверинцев и др. М., 1989.
14. Никитин Е.П., Харламенкова Н.Е. Феномен человеческого самоутверждения. СПб.: Алетейя, 2000.
15. Ханчук Н.Н. Некоторые актуальные проблемы адаптации студентов в процессе обучения в высшей школе// Проблемы социальной адаптации различных групп населения в современных условиях. Владивосток: Изд-во Дальневосточного ун-та, 2000. С. 265.
16. Милославова А.И. Адаптация как социально психологическое явление // Социальная психология и философия. Л.: Наука, 1973. Вып. 2. С.13-21.
17. Кураторская деятельность преподавателя ВУЗа: день за днем/ Под.ред. Н.И. Тихоненкова. – Волгоград: Изд-во «Перемена», 2004.
18. Кон И.С. Психология юношеского возраста: Проблемы формирования личности /Уч. пособие для пед. институтов/ М., 1976. 175 с.
19. Шпрангер Э. Два вида психологии // Хрестоматия по истории психологии / Под ред. П. Я. Гальперина, А. Н. Ждан. М.: Изд-во МГУ, 1980. С. 286—300.
20. Диаграмма связей. Статья из Википедии. [Электронный ресурс]: URL: http://ru.wikipedia.org/wiki/Интеллект-карты.
21. Дубенский Ю.П. Прогнозирование, проектирование и моделирование в социальной работе (Курс лекций. Специальность «Социальная работа»)/ Кафедра педагогики и психологии ОмГУ. 2007.

22. Коган Л.Н., Панова С.Г. Социальное проектирование: его специфика, функции и проблемы// Методологические аспекты социального прогнозирования. Красноярск, 1981.
23. Прусов В.П. Социальные и психологические проблемы целевой интенсивной подготовки специалистов. Ленинград, 1989.
24. Цирельникова Н.А. Физиология и патология механизмов адаптации человека. Новосибирск, 1977.
25. Адаптация и управление свойствами организма. Под ред. Сорокина А.П., Стельникова Г.В., Вазина А.Н. М.: «Медицина», 1977.
26. Адаптация человека. Под ред. Варабашевой В.И., Лихницкой И.И. Ленинград: Издательство «Наука», 1972.
27. Александровский Ю.А. Состояние психической дезадаптации и их компенсация. М., 1976.
28. Дубровина И.В. Практическая психология образования. М., 2000.
29. Лукьяненко Т.И. Практикум по психодиагностике. Г.-А., 2001.
30. Никитина Н.Н., Кислинская Н.В. Введение в педагогическую деятельность. Теория и практика. М., 2004.
31. Основы физиологии человека. Учебник для высших учебных заведений. Том 2. Под ред. академика РАМИН Б.И. Ткаченко. СПб., 1994.
32. Добреньков В.И. Кравченко А.И. Методы социологического исследования. М.: ИНФРА-М, 2004.
33. Котляров И.В. Теоретические основы социального проектирования. Минск,1989.
34. Пасынкова Н.Б. Связь уровня тревожности подростков с эффективностью их интеллектуальной деятельности // Психологический журнал, т.17. 1996.
35. Прохоров А.О. Психические состояния и их проявления в учебном процессе. Казань, 1991.
36. Изард К. Эмоции человека. М.: МГУ, 1980.
37. Орлов А.И. Теория принятия решений /Учебное пособие. М.: «Март», 2004.
38. Литвак Б.Г. Экспертные оценки и принятие решений. М., 1996.
39. Луков В.А. Социальная экспертиза / Ин–т молодёжи. М., 1996.
40. Дятченко Л.Я., Иванов В.Н. и др. Социальные технологии. М., 1995.
41. Горкин А.П. и др. Социальная энциклопедия. М., 2000.
42. Айзенк Х. Психологические теории тревожности: В кн. Тревога и тревожность / Под ред. В.М. Астапова. - СПб.:Питер, 2001.

Математические структуры и моделирование

Выпуск 21

Журнал

Редактор С.Н. Дрозд

Подписано в печать 20.05.2010.
ОП. Формат 60 × 84 1/8. Печ.л. _____ . Уч.-изд.л. _____.
Тираж 100 экз.

Отпечатано в _____
_____, г. Омск, _____, тел. _____
Лицензия _____