МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
им. М.В. ЛОМОНОСОВА

Факультет вычислительной математики и кибернетики

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 9

Под общей редакцией
чл.-корр. РАН Л.Н. Королева

МАКС ПРЕСС

МОСКВА – 2008

ББК 22.19
П75

Печатается по решению
Редакционно-издательского совета факультета
вычислительной математики и кибернетики МГУ имени М.В. Ломоносова

П75 **Программные системы и инструменты:** Тематический сборник/ Под ред. Королева Л.Н. – М: Издательский отдел факультета ВМиК МГУ (лицензия ИД №05899 от 24.09.2001г.); МАКС Пресс, 2008. – № 9. – 220 с.
ISBN 978-5-89407-352-1
ISBN 978-5-317-02659-2

В данный сборник включены научные работы и сообщения, связанные с общими вопросами программирования и информатики; касающиеся разработок и исследований по обработке экспериментальных данных, в частности, по обработке данных ЭЭГ; касающиеся некоторых вопросов математической лингвистики; а также связанные с описанием некоторых инструментальных систем, которые могут оказаться полезными во многих практических приложениях.

В этих публикациях нашли отражение исследования и разработки в области создания программных систем, выполненных учеными, аспирантами и студентами факультета. Большая часть результатов доложена на Ломоносовских Чтениях 2008 года.

Статьи сборника будут интересны студентам, аспирантам и специалистам в области разработки прикладных программных систем с использованием новых информационных технологий.

УДК 519.6+517.958
ББК 22.19

ISBN 978-5-89407-352-1
ISBN 978-5-317-02659-2

От редколлегии:

В предлагаемом читателю тематическом сборнике публикуются статьи, посвященные описаниям инструментальных программных систем, разработанных авторами публикаций, относящихся к методам обработки экспериментальных данных, математической лингвистике, проблемам сетевой обработки, некоторым вопросам теоретического программирования.

В нем также публикуются статьи и заметки общего характера, касающиеся разделов информатики, тесно связанных с программированием.

СОДЕРЖАНИЕ

Раздел I

Общие вопросы программирования и информатики

Рябов Г.Г. (НИВЦ МГУ)

Кодирование состояний Марковских цепей в динамике примитивных триангуляций R^3 и R^4

Решеточные модели и симплициальные комплексы продолжают играть важную роль в теоретической физике и интерес к ним возрос, особенно в последние годы, в связи с методами динамической триангуляции в построении квантовой модели гравитации [2-5]. Кусочно-линейные (PL) комплексы и бизвездные (bistellar) преобразования с появлением нового поколения суперкомпьютеров стали предметом и инструментом вычислительных методов в комбинаторной геометрии и топологии.[6-8] В предлагаемой статье рассматриваются случайные «перестройки» (flips) примитивной триангуляции в $\mathbf{R^3}$ (с вершинами $\mathbf{Z^3}$) как Марковские цепи и исследуются их свойства периодичности, разложимости и эргодичности, тем самым устанавливается асимптотическое поведение триангулированного пространства в целом. Предложены близкие методы для примитивных триангуляций в $\mathbf{R^4}$.

Узловым моментом для компьютерной реализации является корректное (в смысле аксиоматики Колмогорова) вычисление переходных вероятностей в Марковской цепи, для чего строится полное множество элементарных событий и борелевских подмножеств на нем с помощью **кодирования триангулированных разверток** $\mathbf{I^3}$ и $\mathbf{I^4}$, которое по существу является изоморфным отображением. В этом отношении статья продолжает линию, развиваемую в [7-10].

Ключевые слова: *примитивная триангуляция, Диофантовы уравнения, Марковские цепи, кодирование триангулированных разверток, спектр вершинных полиэдров примитивно триангулированных R^3 и R^4 в статистике Бозе-Эйнштейна.*

Случай R^3

Рассматривается триангуляция $\mathbf{R^3}$ с вершинами в целых точках $\mathbf{Z^3}$ и множеством V_1 ребер, коллинеарных примитивным векторам с

максимальным модулем координат равным 1. По сравнению с определенной в [1] примитивной триангуляцией для $\mathbf{R}^n$, как триангуляцией, при которой все симплексы имеют объем 1/n!, рассматриваемый случай расширен, в частности для $\mathbf{R}^3$ включением разбиения единичного куба на 4 симплекса объемом 1/6 и один симплекс объемом 1/3.

Вначале на единичном кубе ($\mathbf{I}^3$) рассмотрим процесс проведения диагоналей во всех его гранях. При проведении в каждой грани одной из двух возможных диагоналей, каждая вершина $\mathbf{I}^3$ будет иметь от 0 до 3-х сходящихся к ней диагоналей. Будем называть это число диагональной степенью вершины (дсв) и обозначать через i. Пусть x_i- число вершин куба с диагональной степенью i. Тогда справедливы соотношения:

$$\Sigma x_i=8;$$
$$\Sigma i x_i=12;\ i=0,1,2,3;$$

Рассматривая эти соотношения как систему Диофантовых уравнений, выпишем все их решения в виде дсв-векторов (x_0, x_1, x_2, x_3):

(0,4,4,0);(0,6,0,2);(1,3,3,1);(2,0,6,0);(2,2,2,2);(2,3,0,3);(3,0,3,2);(3,1,1,3);(4,0,0,4).

Учитывая дополнительные ограничения (сумма диагональных степеней вершин, инцидентных одному ребру больше или равна 2 и меньше или равна 4, а сумма диагональных степеней вершин, образующих каждую грань равна 6), решения (2,3,0,3);(3,0,3,2);(3,1,1,3) не соответствуют никаким способам проведения диагоналей в гранях и поэтому исключаются из дальнейшего рассмотрения. Естественно, что триангуляции куба, соответствующие различным дсв-векторам не конгруэнтны.

Для дсв-векторов (0,6,0,2);(1,3,3,1);(2,0,6,0);(2,2,2,2);триангуляции куба завершаются проведением большой диагонали куба в плоскости, рассекающей куб на 2 треугольные призмы.(рис.1) Все эти триангуляции из 6 симплексов, каждый объемом 1/6.

Для дсв-вектора (4,0,0,4) триангуляция разбивает куб на тетраэдр со стороной $\sqrt{2}$ (объем равен 1/3) и 4 пирамиды (объем каждой 1/6).(рис.1)

Для дсв-вектора (0,4,4,0) не существует никаких «полных» триангуляций куба, т.е. представления куба как нормального симплициального комплекса. Возможны варианты частичной триангуляции, когда 1/3 объема куба есть симплициальный комплекс (из двух симплексов), а 2/3 объема-нет.(Рис.1)

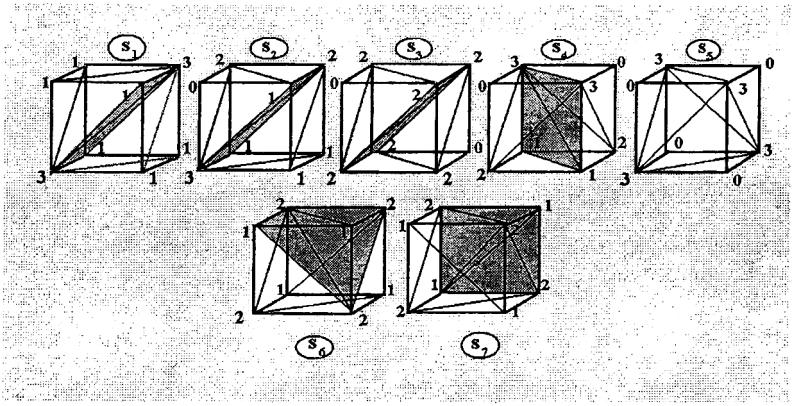


Рис.1 Пять типов полной триангуляции и два типа неполной триангуляции I^3. Рядом с вершинами указаны диагональные степени вершин

Поскольку в дальнейшем будут рассматриваться все возможные положения диагоналей в гранях, то и эти «неполные» триангуляции будут учитываться. Будем обозначать типы триангуляций соответствующие дсв-векторам как s_1,s_2,s_3,s_4,s_5 (полно триангулированы) и s_6,s_7(неполно триангулированы).

Для дальнейшего рассмотрения будем считать, что для каждого единичного куба в $\mathbf{R}^3$ задана одна и та же развертка, в которой задана единая нумерация граней f1,f2,..f6. В случае проведения каким-то способом диагоналей в гранях куба условимся диагональ, идущую в одном направлении обозначать через 0, а в другом-1. Тогда шестиразрядный двоичный код однозначно соответствует каждому из возможных способов проведения диагоналей в гранях. Значение (0 или 1) i-го разряда кода соответствует направлению диагонали в грани fi. Изменение направления диагонали в грани fi приводит к изменению значения в i-ом разряде с 0 на 1 или, наоборот, с 1 на 0, что соответствует сложению 1 со значением разряда по модулю 2. Таким образом есть 2^6=64 всех способов положений диагоналей в гранях, каждый из которых соответствует одному из типов s_i. Ниже приведены данные по числу кодов соответствующих s_i

s_1	s_2	s_3	s_4	s_5	s_6	s_7
4	24	4	12	2	9	9

Выбор наудачу одного из этих 64 кодов можно интерпретировать как случайный выбор одного из единичных кубов случайно гранетриангулированного $\mathbf{R}^3$. Тогда вероятности, что этот куб окажется

полно триангулированным P(+) и неполно триангулированным P(-) равны:

$$P(+) = P(\ U\ s_i; i = 1\text{-}5) = 23/32;$$
$$P(-) = P(s_6, s_7) = 1 - P(+) = 9/32;$$

Пусть задана некоторая триангуляция граней единичного куба, как один из 6-разрядных двоичных кодов, соответствующий типу s_i , и случайно меняется направление диагонали в одной из граней. Будем называть это единичной диагональной перестройкой. Тогда в принципе возможны два варианта-изменение типа s_i или сохранение того же типа. Для дальнейшего рассмотрения поставим в соответствие каждому s_i типу i-ое состояние системы в дискретные моменты времени, связанные с диагональными перестройками и переводящими систему в одно из своих состояний $\{s_1, s_2, \ldots s_7\}$. Таким образом, для полного и корректного задания цепи Маркова необходимо определить матрицу переходных вероятностей. Считая равновероятным изменение направления диагонали в любой грани, можно определить все множество элементарных исходов для каждого из типов s_i при изменении в каждой грани. Для однозначности достаточно задать на некоторой зафиксированной развертке I^3 номера граней, которые совпадают с номерами двоичных разрядов в 6-разрядном коде, и поставить в соответствие направлениям диагоналей 0 и 1. Каждому элементарному исходу при изменении положения диагонали в одной грани однозначно соответствует пара шестиразрядных двоичных кодов к1,к2, где к1 - код исходного положения диагоналей, а к2-код получившегося положения после перестройки.(Рис.2а) Аналогично кодируется 3d развертка для I^4 (рис.2b).

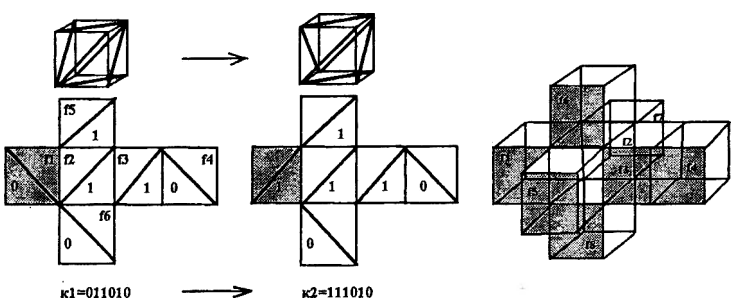
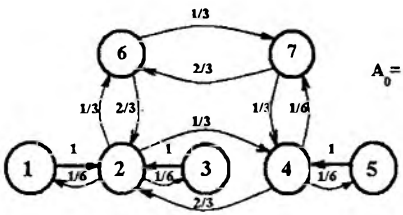


Рис.2.а)Триангуляции I^3, связанные изменением диагонали в одной грани, соответствующие развертки для кодирования (грань изменения затемнена) и коды триангуляций. Номера граней соответствуют номерам разрядов в кодах.б) 3d-развертка I^4

На множестве всех элементарных исходов (пар кодов) Ω определяются переходные вероятности Марковской цепи-p_{ij}, а затем изучаются свойства таких Марковских цепей (возвратность, периодичность, разложимость и эргодичность) при различном числе одновременно изменяемых в один момент дискретного времени граней в I^3. Поэтому ниже рассматривается несколько вариантов и дается краткая характеристика процесса для каждого варианта.

1. Рассматриваются все 7 состояний системы, в каждый дискретный момент времени случайно меняется диагональ в одной из граней. На рис.2 показаны состояния системы и переходные вероятности между состояниями. Рядом приведены: матрица переходных вероятностей за один шаг по времени, матрицы для четных и нечетных степеней для больших n.Цепь возвратная, периодическая.



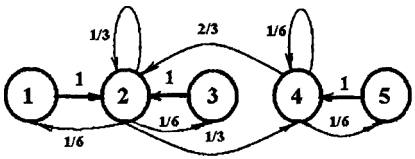
$$A_0 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1/6 & 0 & 1/6 & 1/3 & 0 & 1/3 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2/3 & 0 & 0 & 1/6 & 0 & 1/6 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 2/3 & 0 & 0 & 0 & 0 & 1/3 \\ 0 & 0 & 0 & 1/3 & 0 & 2/3 & 0 \end{pmatrix};$$

$$A_0^{2n+1} = \begin{pmatrix} 1/8 & 0 & 1/8 & 3/8 & 0 & 3/8 & 0 \\ 0 & 3/4 & 0 & 0 & 1/16 & 0 & 3/16 \\ 1/8 & 0 & 1/8 & 3/8 & 0 & 3/8 & 0 \\ 1/8 & 0 & 1/8 & 3/8 & 0 & 3/8 & 0 \\ 0 & 3/4 & 0 & 0 & 1/16 & 0 & 3/16 \\ 1/8 & 0 & 1/8 & 3/8 & 0 & 3/8 & 0 \\ 0 & 3/4 & 0 & 0 & 1/16 & 0 & 3/16 \end{pmatrix}; \quad A_0^{2n} = \begin{pmatrix} 0 & 3/4 & 0 & 0 & 1/16 & 0 & 3/16 \\ 1/8 & 0 & 1/8 & 3/8 & 0 & 3/8 & 0 \\ 0 & 3/4 & 0 & 0 & 1/16 & 0 & 3/16 \\ 1/8 & 0 & 1/8 & 3/8 & 0 & 3/8 & 0 \\ 0 & 3/4 & 0 & 0 & 1/16 & 0 & 3/16 \\ 1/8 & 0 & 1/8 & 3/8 & 0 & 3/8 & 0 \end{pmatrix};$$

Рис.3. Схема и матрица переходных вероятностей A_0 и матрицы для четных и нечетных моментов дискретного времени

2. Во всех последующих вариантах рассматриваются 5 состояний системы, соответствующие полным триангуляциям (s_1,s_2,s_3,s_4,s_5). В каждый дискретный момент времени может случайно измениться диагональ в одной из граней с условием, что это не переводит куб в состояние неполной триангуляции (s_6,s_7). Если возможное изменение именно такое, то в этот момент система остается в предыдущем состоянии (на диаграммах это графически изображено петлей). Ниже приведены результаты для случаев, когда в дискретный момент времени случайно меняются h граней куба (1,2,...6).

h=1; Матрицы переходных вероятностей на рис.4. Цепь эргодическая.

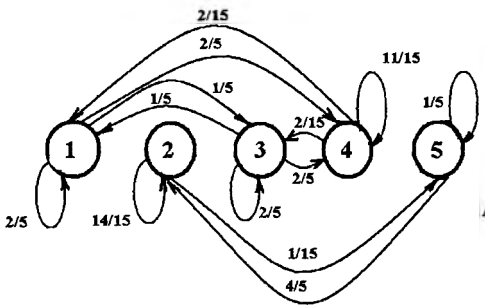


$$A_1=\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1/6 & 1/3 & 1/6 & 1/3 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 2/3 & 0 & 1/6 & 1/6 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$A_1^n=\begin{pmatrix} 0.0870 & 0.5217 & 0.0870 & 0.2609 & 0.0434 \\ 0.0870 & 0.5217 & 0.0870 & 0.2609 & 0.0434 \\ 0.0870 & 0.5217 & 0.0870 & 0.2609 & 0.0434 \\ 0.0870 & 0.5217 & 0.0870 & 0.2609 & 0.0434 \\ 0.0870 & 0.5217 & 0.0870 & 0.2609 & 0.0434 \end{pmatrix}$$

Рис.4. Случайное изменение в одной грани (h=1)

h=2;Цепь распалась на 2 цепи с состояниями (1,3,4)(2,5),каждая из которых эргодическая.(рис.4)

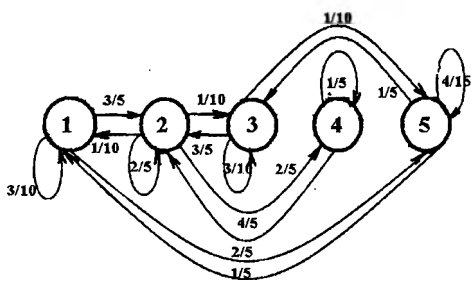


$$A_2=\begin{pmatrix} 2/5 & 0 & 1/5 & 2/5 & 0 \\ 0 & 14/15 & 0 & 0 & 1/15 \\ 1/5 & 0 & 2/5 & 2/5 & 0 \\ 2/15 & 0 & 2/15 & 11/15 & 0 \\ 0 & 4/5 & 0 & 0 & 1/5 \end{pmatrix}$$

$$A_2^n=\begin{pmatrix} 0.2 & 0 & 0.2 & 0.6 & 0 \\ 0 & 0.9230 & 0 & 0 & 0.0770 \\ 0.2 & 0 & 0.2 & 0.6 & 0 \\ 0.2 & 0 & 0.2 & 0.6 & 0 \\ 0 & 0.9230 & 0 & 0 & 0.0770 \end{pmatrix}$$

Рис.5. Случайное изменение в двух гранях (h=2)

h=3;Цепь эргодическая.(рис.6)

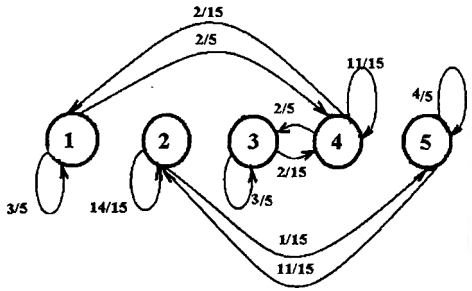


$$A_3=\begin{pmatrix} 3/10 & 3/5 & 0 & 0 & 1/10 \\ 1/10 & 2/5 & 1/10 & 2/5 & 0 \\ 0 & 3/5 & 3/10 & 0 & 1/10 \\ 0 & 4/5 & 0 & 1/5 & 0 \\ 1/5 & 0 & 1/5 & 0 & 3/5 \end{pmatrix}$$

$$A_1^n=A_3^n=A_5^n;$$

рис.6. Случайное изменение в трех гранях (h=3)

2.4.h=4;Цепь распалась на 2 цепи.(рис.7)



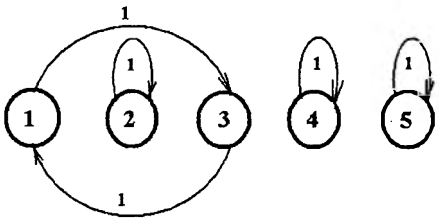
$$A_4 = \begin{pmatrix} 3/5 & 0 & 0 & 2/5 & 0 \\ 0 & 14/15 & 0 & 0 & 1/15 \\ 0 & 0 & 3/5 & 2/5 & 0 \\ 2/15 & 0 & 2/15 & 11/15 & 0 \\ 0 & 12/15 & 0 & 0 & 3/15 \end{pmatrix}$$

$$A_4^n = A_2^n \ ;$$

Рис.7. Случайное изменение в четырех гранях (h=4)

2.5.h=5;Цепь эргодическая, т.к. $A_1=A_5$ и поэтому $A_1^n=A_5^n$.

2.6.h=6;(Полная инверсия диагоналей в гранях). Диаграмма цепи и матрица переходных вероятностей приведены на рис.8.



$$A_6 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Рис.8. Изменение диагоналей во всех шести гранях

Обобщением приведенных результатов могут служить следующие **утверждения**.

1. Случайные перестройки диагоналей в гранях примитивной триангуляции $\mathbf{R}^3$ индуцируют Марковскую цепь с конечным числом состояний и дискретным временем.
2. При допущении всех типов триангуляции граней и при одиночной перестройке в каждый момент времени процесс является периодическим.
3. При допущении только тех типов триангуляции граней, которым соответствует полная триангуляция куба (s_1,s_2,s_3,s_4,s_5) **процесс для нечетного числа меняющихся граней является эргодическим с одинаковым стационарным распределением. Для четного числа меняющихся граней процесс распадается на подцепи с эргодическими свойствами.**

В трехмерном случае методом двоичного кодирования двумерных граней можно перечислить все варианты вершинных полиэдров при примитивной триангуляции на множестве ребер V_1, контролируя и отбрасывая при перечислении коды, не соответствующие полным нормальным триангуляциям. Поскольку рассматриваются 8 октантов (кубов) с общим числом граней 36, общее число всех вариантов расположения диагоналей в них равно 2^{36}.Расчеты на компьютере дали следующие результаты. Каждый допустимый код отображался в λ- вектор ($\lambda_1.,\lambda_2....$),где λ_i-соответствует i-ому кубу и равен номеру типа триангуляции ($s_1,s_2,...s_5$).Обозначим это множество через Λ. Это множество содержит и вектора, соответствующие конгруэнтным триангуляциям. Определенную характеристику этого множества можно получить, проведя следующую группировку. Для этого определим множество M таких μ-векторов ($\mu_1,\mu_2,...\mu_5$), где μ_k-число кубов с триангуляцией k-го типа и $\Sigma\mu_k=8$. Тогда каждому λ-вектору соответствует некоторый μ-вектор, а каждому μ-вектору может соответствовать один ,несколько λ-векторов или вообще ни одного λ-вектора. При этом для $\mu_1\neq\mu_2$ соответствующие им триангуляции заведомо не конгруэнтны.

Число μ-векторов в нашем случае равно числу размещения 8 частиц в 5 ящиках без ограничений (статистическая модель Бозе-Эйнштейна) и равно (8+5 -1)!/8!4!= 495. Если расположить все μ-вектора в лексикографическом порядке вдоль оси x, а по оси y откладывать число λ-векторов (логарифмическая шкала), соответствующих каждому μ-вектору, то получим результат, изображенный на рис.8 ,который можно рассматривать как спектр частот неконгруэнтных вершинных полиэдров при примитивной триангуляции $\mathbf{R}^3$. Отметим, что для μ-векторов (1,0,1,0,6),(1,0,7,0,0),(3,0,5,0,0),(5,0,3,0,0), (7,0,1,0,0) не существует ни одной полной триангуляции. Для μ-векторов (0,5,0,3,0), (0,5,1,2,0),(0,6,0,2,0),(1,5,0,2,0) число различных полиэдров равно максимальному числу 188116992.Близкий к фрактально-периодическому характеру спектра вызван с одной стороны порядком расположения μ-векторов, с другой стороны коррелирует с числом типов s_2 и s_3 («минимально симметричных» среди s_1,s_2,s_3,s_4,s_5).

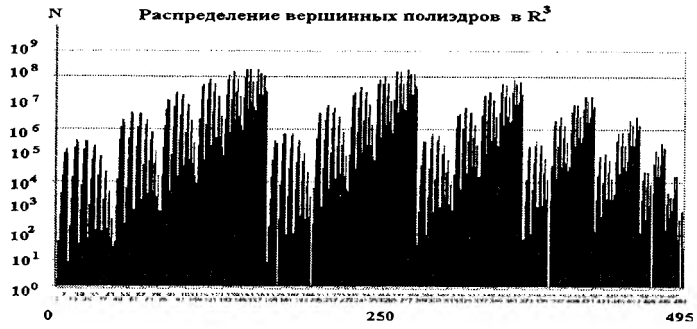


Рис.9. Распределение вершинных полиэдров при примитивных триангуляциях R^3

Случай R^4

Примитивная полная триангуляция единичного куба в $\mathbf{R}^4$ естественно однозначно определяет полную примитивную триангуляцию в каждой его трехмерной грани, которая может быть одной из 5 типов, определенных в предыдущем разделе. Так что в случае перечисления всех неконгруэнтных типов полной примитивной триангуляции в $\mathbf{R}^4$ можно рассматривать все возможные восьмизначные (по числу трехмерных граней в $\mathbf{I}^4$) пятиричные числа(по числу типов) с проверкой «совместимости» диагоналей в двумерных гранях. При этом требуется учесть все варианты ориентации трехмерных граней.

Однако воспользуемся более прозрачной схемой перечисления из предыдущего раздела, т.е. будем рассматривать все возможные положения диагоналей в двумерных гранях $\mathbf{I}^4$. Тогда для некоторой общей для всех случаев развертки $\mathbf{I}^4$ перенумеруем все двумерные грани (их 24) и этот номер будет номером двоичного (по числу положений диагонали в двумерной грани) разряда в 24-разрядном числе. Таким образом число всех возможных вариантов проведения диагоналей в двумерных гранях $\mathbf{I}^4$ равно 2^{24}. Для каждого такого кода надо проверить 1).полна ли триангуляция для всех трехмерных граней. Совпадение диагоналей в двумерных гранях, общих для трехмерных граней, реализуется автоматически, поскольку такая грань представлена одним разрядом.

Соответствующая программа была разработана и были перечислены все типы полных примитивных триангуляций $\mathbf{I}^4$ в виде векторов $(\lambda_1,\lambda_2,\ldots\lambda_8)$, где λ_k равно i как номеру типа триангуляции данной (k-ой) трехмерной грани (i=1,2,…5). Таких различных λ-векторов оказалось 2494 из общего числа 2^{24}.Обозначим это множество через Λ.

Как и в предыдущем разделе отобразим множество Λ на множество M

Проведенные компьютерные расчеты дали следующие результаты.

IΛI=2494;IMI=495; Число различных μ , которым соответствует хотя бы одна примитивная триангуляция $\mathbf{I}^4$, равно 295. Частоты наборов изменяются в диапазоне от 0 до 58. Спектр частот наборов при лексикографическом упорядочении векторов из M по оси x. (рис.10)Принципиально можно применить методы анализа свойств Марковских процессов из предыдущего раздела, хотя это связано с определенными техническими трудностями. Кратко характеризуем их.

1.Для вычисления переходных вероятностей в системе из 2494 состояний с изменением k диагоналей в 24 гранях потребуется C 2494 2^{24}=C 10^{11} (C-некоторая константа) операций.

2.Одноразовое перемножение матриц 2494 порядка потребует 10^{11}-10^{12} операций, а при исследовании эргодичности и многократном умножении примерно- 10^{13}-10^{14} операций.

Таким образом, речь может идти о расчетах на суперкомпьютерах, при высокой распараллеливаемости задачи, для которой здесь есть все основания.

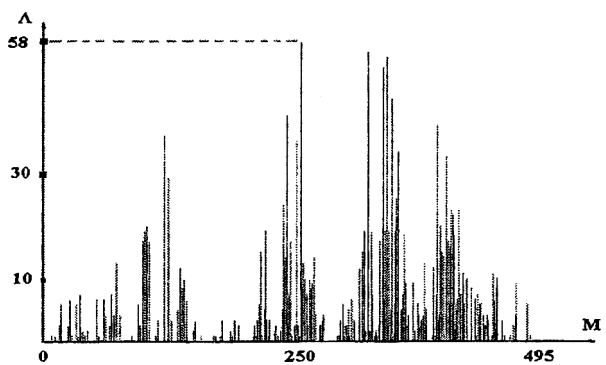


Рис.10. Распределение вершинных полиэдров при примитивных триангуляциях R^4

Дискуссия

1.Методы отображения примитивных триангуляций в двоичные коды дают возможность в ряде случаев($\mathbf{R}^3$,$\mathbf{R}^4$) перечислить на компьютере все элементарные исходы перестроек и вычислять точно

переходные вероятности, а на этом основании устанавливать основные свойства Марковских процессов.

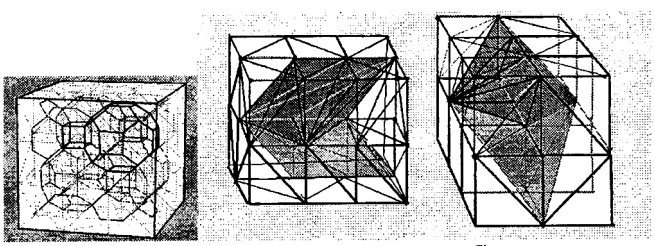
2.Примитивные триангуляции и их перестройки как Марковские процессы могут быть моделью для динамики триангуляций более общего вида, в частности динамических триангуляций в квантовых моделях (3d+1)размерной гравитации.[2]

3.Перестройки триангуляций включены в инструментальную систему «топологический процессор»[7-11] как макрооперации.

4.Результаты перечисления позволяют представить дуальную картину триангуляций в виде соответствующих вершинных многогранников. Пример такой дуализации для λ-вектора (0,0,0,0,8) представлен на рис.11а.

5. Предложенное кодирование дает возможность представить реальные границы применения таких методов для современных суперкомпьютеров кластерного типа. Так для перечисления всех возможных примитивных триангуляций в $\mathbf{I}^5$ подобными методами потребуется рассмотреть 2^{80} вариантов, поскольку для $\mathbf{I}^5$ число двумерных граней равно 80. Это число вариантов примерно равно 10^{24} ,что далеко за реальной производительностью современных компьютерных систем. Однако если ограничить число рассматриваемых типов до 2-х из {s1,...s5} типов трехмерной триангуляции, то для $\mathbf{I}^5$ число всех возможных типов можно оценить как 2^N, где $N=C_5^3 2^2 =40$,т.е. число рассматриваемых вариантов не более 10^{13}, что делает компьютерные расчеты реальными.

6.Расширение множества Vp (p=1,2,3...) при примитивных триангуляциях $\mathbf{R}^3$,$\mathbf{R}^4$ можно рассматривать как одновременные перестройки двух или более двумерных граней вместе с их смежными ребрами (в том числе и единичными). Так на рис.11б показано образование симплициального комплекса с примитивными ребрами из V_2 и V_3, не затрагивающее общей триангуляции на V_1 вне параллелепипеда из 3x2x2 кубов.



а) б)

Рис.10. а) Дуальные многогранники для λ=(0,0,0,0,8); б) Локальная перестройка в примитивной триангуляции $\mathbf{R}^3$ с включением ребер из V_2 иV_3

Заключение

Рассмотрение Марковских процессов может служить инструментом оценки статистической картины поведения, при случайных перестройках в гранях, примитивно триангулированных пространств $\mathbf{R^3}$, $\mathbf{R^4}$, в том числе и **метрики,** соответствующей кратчайшим путям по ребрам таких триангуляций. Так применение методов двоичного кодирования к определению кратчайших путей в случайном примитивно триангулированном $\mathbf{R^3}$ позволило на суперкомпьютере «СКИФ МГУ Чебышев» (60 терафлопс) рассчитывать метрику Хаусдорфа между подмножествами на 3-х мерных решетках с числом вершин 10^{10} и числом ребер 10^{13}-10^{14} за десятки минут машинного времени (при параллельной работе 128 процессоров и 256 Гбайт оперативной памяти).

Автор приносит благодарность Л.Н.Королеву, А.В.Тихонравову, Б.Н.Четверушкину за внимание и обсуждения тематики работы.

Литература

1.E.Steingrimsson. «Permutation statistics of indexed and poset permutations.» Ph.d.thesis MIT(1992).

2.S.Negami. «Diagonal flips of triangulations on surfaces».Yokohama Math.Journal.v.45 №2.1998

3.В.А.Малышев. Вероятность вокруг квантовой гравитации: планарная гравитация. //УМН n.54 №4(328) 1999.

4.P.Collet, J-P.Eckman. «Dynamics of triangulations.» arXiv:math-ph/0412085v1,23 Dec 2004

5.J.Ambjorn, J.Jurkevich, R.Loll. «Reconstructing the Universe.» Phys.Rev.D72(2005)
(arXiv:hep-th/0505154v2)

6.В.М.Бухштабер, Т.Е.Панов. Торические действия в топологии и комбинаторике. МЦНМО.(2004)

7.Г.Г.Рябов. «Алгоритмические основы топологического процессора.» Труды научной конференции МСО-2005,53-58(2005)

8.Г.Г.Рябов, В.А.Серов. «Отображения целочисленных множеств и евклидовы приближения».//Вычислительные методы и программирование. МГУ.2007 т.8,№1.10-19

9.Г.Г.Рябов. «О путевом кодировании k-граней в n-кубе»//Вычислительные методы и программирование. МГУ.2008.т.9.№1.20-22.

10.G.Ryabov, V.Serov. «Simplicial-lattice model and metric-topological constructions.» Proceedings of PRIP-2007,2,135-140(2007).

11.Г.Г.Рябов, В.А.Серов. «Компьютерные комбинаторно-топологические построения и их преобразования».//Информационные технологии и вычислительные системы. РАН. 2008. №2 .69-80

Смелянский Р.Л., Шалимов А.В.

Разработка системы компактного представления программ

Введение

Проблема компактного представления программ является актуальной, особенно для встроенных систем управления реального времени [1,2]. В таких системах, как правило, критическим ресурсом является оперативная память, т.к. внешняя память у них обычно отсутствует. Ограничения по памяти проистекают из двух основных причин: ограничения массогабаритных параметров и ограничения по энергопотреблению. В этих условиях важной характеристикой программы опять (как и в начале развития ЭВМ) стал необходимый программе объем оперативной памяти. Вышесказанное объясняет важность методов компактного представления программ (методы КПП) в современных условиях.

В статье описана система компактного представления последовательных программ, реализующая метод КПП на основе частотных характеристик их поведения. Приведены результаты теоретических и экспериментальных исследований, которые показывают потенциальную возможность применения предложенного метода компактного представления программ в бортовых вычислительных системах. В первой части описана задача, в рамках которой разрабатывался предложенный метод КПП. Во второй части дано общее описание предложенного метода КПП. В третьей части приведены выведенные математические зависимости, позволяющие для заданной бортовой вычислительной системы определить возможность применения предложенного метода КПП. В четвертой части описан предложенный метод определения частотных характеристик программы. В пятой части идет речь о способах определения редко выполняемого кода. В шестой части описаны результаты экспериментальных исследований разрабатываемой системы КПП.

1. Описание задачи

Данный метод КПП разрабатывается для применения в бортовых вычислительных системах (БВС). Это связано с тем, что в большинстве используемых сейчас бортовых цифровых вычислительных машинах доступно ограниченное количество оперативной памяти. Например, объем памяти для БВС самолетов находится в диапазоне от 0.5 до 4.5 МБ [3]. Применение методов КПП позволит расширить

функциональные возможности БВС. Кроме того, современные подходы к проектированию и реализации программного обеспечения для БВС предполагают использование низкоуровневого программирования в большом объеме и отказ от языков программирования высокого уровня. Причина в том, что большинство систем программирования на ЯВУ порождает громоздкий по памяти код.

Перечислим особенности БВС, которые надо учитывать при применении методов КПП.

1. Повышенные требования к надежности и безопасности функционирования.
2. Работа в режимах жесткого реального времени. В БВС не допустим выход времени выполнения программы за директивные сроки.
3. Ограничение по памяти. БВС характеризуются малым количеством основной памяти.
4. Жесткие ограничения на массогабаритные параметры системы. Появление дополнительных аппаратных блоков ведёт к увеличению габаритов, массы и энергопотребления.

Тогда для применения в БВС любой метод КПП должен удовлетворять следующим условиям применимости в БВС:

1. Время выполнения сжатой программы не должно превышать время выполнения исходной более чем в заданное число раз. Заданный коэффициент нужен для использования зазора между фактическим временем выполнения программы и директивным сроком её выполнения.
2. Накладные расходы по памяти при использовании метода КПП не должны превышать суммарного выигрыша по памяти от сжатия набора программ БВС.

Из проведенного во время исследовательской работы обзора методов КПП и анализа их применимости в БВС[5] были сделаны следующие выводы:

1. Возможность применения в БВС методов КПП без процесса распаковки.
2. Методы КПП, использующие процесс распаковки, обладают более высоким коэффициентом сжатия, чем методы КПП без распаковки. Но применение в БВС *существующих* методов КПП с распаковкой невозможно из-за сильного увеличения времени выполнения скомпактированной программы.

В связи с этим актуальна разработка нового метода КПП с распаковкой, при которой оставались бы высокими возможности по

сжатию с учетом специфики БВС. Заметим, что декомпрессор должен быть программным, т.к. аппаратный декомпрессор потребует увеличения массогабаритных параметров БВС. В результате анализа был предложен описанный в данной работе метод.

2. Общее описание предложенного метода

Идея предложенного метода восходит к работам [6,7,8] и основана на двух фактах. Первое: в последовательной программе исполнение 15-20% кода программы занимает 80% времени её исполнения [6,7]. Второе: программа в интерпретируемой форме, как правило, занимает меньше места, чем в откомпилированной [8]. На основе этих двух фактов было предложено исследовать метод, который заключается в том, что редко выполняемые фрагменты кода программы [10] хранятся в сжатой интерпретируемой форме и динамически, по мере необходимости, распаковываются и выполняются, а часто выполняемый код компилируется.

Исследуемый метод компактного представления программы состоит из двух основных частей: сжатие программы и выполнение сжатой программы. Рисунок 1 показывает основные принципы работы метода.

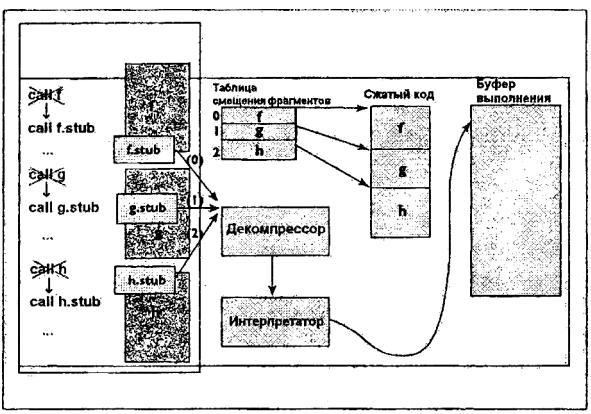


Рисунок 1

Рассмотрим программу с тремя редко выполняемыми фрагментами кода *f*, *g* и *h*, как показано в левой части рисунка 1. Структура метода и организация сжатого кода показана в правой части рисунка 1. Каждый выбранный фрагмент кода программы при её сжатии заменяется в программе очень короткой последовательностью команд - заглушкой, которая вызывает декомпрессор, чья работа

заключается в распаковке интерпретируемого кода этого фрагмента в буфер выполнения и передаче управления интерпретатору, который будет выполнять распакованный код. Таблица смещения фрагментов определяет, где внутри сжатого кода начинается код для данного фрагмента. Заглушка для каждого фрагмента передает декомпрессору аргумент, который является индексом в этой таблице; этот аргумент показан меткой ((0),(1),...) на дуге от каждой заглушки до декомпрессора. Декомпрессор использует этот аргумент для индексации в таблице смещения фрагментов, извлекает начальный адрес сжатого кода соответствующего фрагмента и начинает генерацию несжатого интерпретируемого кода в буфер выполнения. Затем декомпрессор передает управление интерпретатору сгенерированного кода. Когда распакованный фрагмент заканчивает свое выполнение, он обычным способом возвращает управление вызывающей программе.

Заметим, что перед применением предложенного метода, к исходной программе можно применить и другие методы КПП, например, оптимизацию по памяти, сжатие библиотек и т.п. Также отметим, что вместо интерпретации можно использовать и другие техники, например, сжатие предварительно скомпилированных фрагментов.

Такая организация метода позволяет управлять степенью сжатия программы в зависимости от требований к скорости выполнения программы и доступного размера памяти. Эта возможность для бортовых вычислительных систем позволяет учитывать директивные сроки выполнения программ.

3. Применение предложенного метода в БВС

В ходе исследовательской работы [5] получены математические зависимости, позволяющие для заданной БВС определить возможность применения исследованного метода. Введем следующие обозначения

- θ - порог - входной параметр разработанного метода КПП. Доля выполняемого кода программы, которая в последствии будет переведена в сжатое интерпретируемое представление.
- $\tau \geq 1$ - коэффициент возможного увеличения времени выполнения программы. Заданный коэффициент нужен для использования «зазора» между фактическим временем выполнения программы и директивным сроком её выполнения.
- $\lambda(\theta)$ - коэффициент сжатия предложенного метода КПП. Определяется экспериментально для конкретной реализации данного метода КПП.
- I - количество машинных команд на выполнение одной команды интерпретируемого кода.

- $M(КПП)$ - дополнительный расход памяти на использование метода КПП в БВС.

Заметим, что последние три параметра – это характеристики конкретной реализации метода. Подробнее об этом будет сказано далее.

Тогда для использования предложенного метода в БВС необходимо:

1. Выбрать параметр $\theta \le \frac{\tau - 1}{I - 1}$.
2. Выбрать программы для сжатия, суммарный объем которых больше $\frac{M(КПП)}{1 - \lambda(\theta)}$.
3. Если условия двух выше описанных пунктов одновременно не могут выполняться или необходим более высокий коэффициент сжатия программы, то возможна необходимость наложения требования об увеличении производительности БВС в $\frac{1 + \theta * (I - 1)}{\tau}$ раз.

Если следовать указанным выше рекомендациям на использование предложенного метода КПП, то гарантируется, что время выполнения набора программ увеличится не более чем θ раз при коэффициенте сжатия равным $\lambda(\theta)$.

4. Метод оценки частот выполнения фрагментов кода последовательной программы

В этом разделе описан метод изменения программы для получения частотных характеристик исполнения её фрагментов.

Постановка задачи

Дан текст последовательной программы (язык программирования в данном случае несущественен). Для каждого входного параметра этой программы известен диапазон его значений и функция распределения значений на этом диапазоне. Предполагается, что программа не содержит не исполняемых фрагментов. Требуется определить средние частоты выполнения каждого фрагмента, т.е. линейного участка исследуемой программы [6,7].

Изменение программы

Программа представляется в виде множества функций и множества входных параметров. Функция – это множество базовых блоков. Базовый блок – это пара (линейный участок, функция перехода). Линейный участок – это последовательность операторов, не

содержащая операторов управления. Функция перехода определяет условие перехода от одного базового блока к другому. Входной параметр программы - это переменная исследуемой программы, через которую в программу передают данные извне. Для подсчета средних частот исполнения преобразуем исходную программу без изменения её основной функциональности.

Программа изменяется следующим образом:

1. В программу добавляются функции инициализации, которые присваивают входным параметрам программы значения, сгенерированные по закону распределения их значений согласно [11,12].
2. Команды чтения из внешних источников (с клавиатуры и т.п.) входных параметров программы заменяются на обращение к функциям инициализации.
3. В начало каждого базового блока добавлены счетчики. Счетчики увеличиваются на единицу каждый раз, когда происходит переход управления к базовым блокам. После окончания работы программы счетчики базовых блоков будут содержать количество передач управления на каждый базовый блок.

Не трудно видеть, что эти преобразования не затрагивают функциональность исходной программы, хотя несколько изменяют её временные характеристики.

Определение средних частот

Общая идея заключается в использовании метода статистических испытания (Метод Монте-Карло) [11]. Этот метод заключается в том, что со случайной величиной проводят опыты, получают значения случайной величины, и на их основе рассчитывают статистические характеристики этой случайной величины.

Произведем N прогонов программы. Способ определения необходимого числа прогонов указан ниже. Получим N значений счетчиков. Заметим, что результаты каждого прогона будут отличаться от результатов предыдущего, поскольку каждый раз при работе программы происходит генерация новых значений входных параметров. Рассмотрим среднее арифметическое значений каждого счетчика базового блока модели, т.е.

$$F_j^* = \frac{1}{N} \times \sum_{i=1}^{N} f_j^i \qquad (1),$$

где f_j^i значение счетчика j-го базового блока после i-го запуска модели. Отметим, что полученные так величины будут соответствовать средней частоте срабатывания команд программы. Покажем, что указанная величина является средней частотой базового блока.

Утверждение. Величина F_j^* (1), равная сумме значений всех счетчиков базового блока на каждом прогоне программы, деленная на N, стремится к средней частоте выполнения базового блока при стремлении числа запусков программы к бесконечности. И среднюю частоту можно вычислить с заранее заданной точностью.

Доказательство теоремы можно посмотреть в [5].

На практике возможно применять следующий алгоритм [11,12] вычисления средний частоты базового блока F_j^* с точностью ε и достоверностью β.

1. Устанавливается счетчик числа прогонов N=0;
2. Проводится один прогон измененной программы. По результату прогона запоминается число выполнений базового блока f_j^i. N = N + 1;
3. Вычисляются $F_j^* = \frac{1}{N} \times \sum_{i=1}^{N} f_j^i$ и $\sigma_{f_j}^{*\,2} = \frac{1}{N-1} \sum_{i=1}^{N} (f_j^i - F_j^*)^2$;
4. Проверяется выполнение условия
5. $N > \left(\frac{\sigma_{f_j}^* * t_\beta}{\varepsilon} \right)^2$, где t_β - квантиль порядка β (вычисляется по таблицам [11, 12]).;
6. Если условие выполнено, то задача определения средней частоты базового блока решена (т.е. F_j^* определена с заданной точностью). Иначе проводится очередной прогон программы (шаг.2).

Заметим, что аналогичным изменением программы можно определить средние значения различных характеристик программы. Например, количество переходов между базовыми блоками, среднее число шагов работы программы и т.п.

Важно отметить, что предлагаемая техника позволяет избежать исследования вопроса об условных и безусловных вероятностях переходов между базовыми блоками. Обычно в работах, где исследовались частотные характеристики программ, явно или не явно делалось предположение, что вероятность перехода к некоторому базовому блоку не зависит от того, по какому пути мы пришли к нему. Такое предположение является весьма спорным. Предложенная здесь техника учитывает зависимости в программе по управлению и их влияние на значения вычисляемых частот.

При таком общем подходе пока не понятно, как заранее определить, сколько раз надо прогнать измененную программу для

достижения указанной точности. Только в процессе работы метода можно увидеть достигнутую точность.

5. Определение редко выполняемых частей кода программы

Дан текст последовательной программы, в которой нет не исполняемых фрагментов. Команды программы равнозначны. Выполнение команды занимает 1 единицу времени. Для каждой команды программы известна частота её выполнения.

Рассмотрим три различные постановки задачи определения редко выполняемого кода программы.

1. **При ограничениях по частоте**. Под редко выполняемым кодом программы понимается совокупность команд программы, частота выполнения которых не превосходит заданного числа K.

2. **При ограничениях по памяти**. Входной информацией является требуемый размер S компилируемой части программы в исследуемом методе. Отбираются базовые блоки в порядке уменьшения их частоты выполнения, пока сумма команд отобранных базовых блоков не превысит S. Тогда все неотобранные блоки считаются редко выполняемыми.

3. **При ограничениях по времени**. Одним из входных параметров метода является T - среднее количество интерпретируемых команд программы. Отбираются базовые блоки в порядке увеличения частоты, пока сумма весов (частота базового блока, умноженное на количество команд) отобранных базовых блоков не превысит T. Тогда все отобранные блоки считаются редко выполняемыми.

В исследуемом методе используется 3-ой способ определения редко выполняемого кода программы.

6. Система компактного представления программ

Напомним, что основная идея исследуемого метода заключается в том, что редко выполняемые фрагменты кода программы хранятся в сжатой интерпретируемой форме и динамически, по мере необходимости, распаковываются и выполняются, а часто выполняемый код компилируется.

Система компактного представления программы, реализующая предложенный метод, написана на языке C++. Она состоит из двух частей (рисунок 2): сжатие программы и выполнение сжатой программы. На вход система получает программу на языке Си, функции распределения входных параметров этой программы и максимальную долю сжимаемого кода программы. На выходе имеем сжатую

программу и файлы с таблицей смещения и сжатым интерпретируемым кодом.

Компрессор представляет функциональность перевода редко выполняемого кода программы в интерпретируемое представление и сжатие его как текста. В компрессоре на основе функций распределения входных параметров происходит определение частот фрагментов кода программы. На основе доли сжимаемого кода и полученных частот происходит определение редко выполняемого кода. Далее происходит группировка редко выполняемых фрагментов кода в области так, чтобы накладные расходы от сжатия этих областей не превышали полученного выигрыша.

Декомпрессор состоит, главным образом, из интерпретатора и функциональности распаковки интерпретируемого представления.

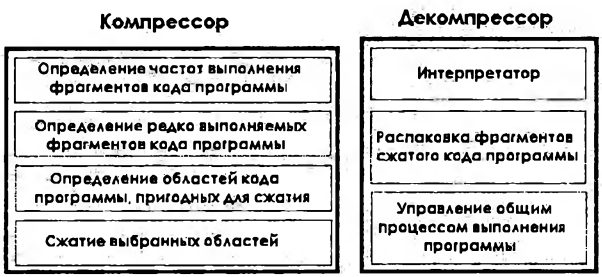


Рисунок 2

Испытания системы КПП проводились на программах, реализующих функциональные задачи, решаемые в БВС современных самолетов. Целью является определение зависимости между входным параметром θ и коэффициентом сжатия метода λ.

В качестве тестовых данных использовались программы БВС из проекта DrTesy лаборатории ВК [9]. Целью этого международного проекта было сравнительное испытание методов и инструментальных средств моделирования и логического анализа программного и аппаратного обеспечения встроенных систем реального времени на практическом примере. В качестве примера встроенной системы исследовался навигационный комплекс летательного аппарата при решении задачи полета по заданному маршруту. Подобранные тестовые данные участвуют в решении следующих задач:

1. Коррекция координат самолета.
2. Полет на очередной и внеочередной пункт маршрута.
3. Маловысотный полет с огибанием рельефа.
4. Горизонтальный полет.

Для каждой тестовой программы производилось 10 запусков системы с разными значениями θ (0.1, 0.2, ... , 1). Для каждого запуска запоминался получаемый коэффициент сжатия λ. После проведения всех экспериментов производилось усреднение полученных значений. В результате была получена зависимость $\lambda(\theta)$, представленная на рисунке 3.

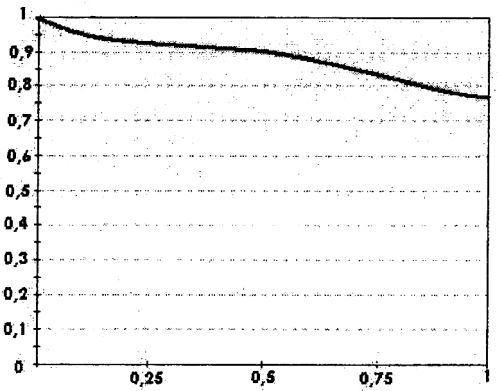


Рисунок 3

Таким образом, при θ = 100% достигается максимальный коэффициент сжатия λ = 77%. При θ = 0.5% достигается коэффициент сжатия λ = 0.89%. При $0.2 < \theta < 0.4$ на тестовых программах происходит уменьшение скорости роста коэффициента сжатия. Отсюда следует рекомендация выбирать входной параметр из указанного диапазона.

Проведя анализ кода программы, реализующей декомпрессор предлагаемого метода, были получены следующие характеристики реализации метода:

1. В среднем на выполнение одной команды интерпретируемого кода происходит выполнение шести машинных команд (I = 6) .
2. Для работы декомпрессора необходимо 100 Кб оперативной памяти БВС (М(КПП) = 100 Кб).

Тогда для целесообразности применимости реализации предложенного метода в БВС необходимо:

1. выбрать параметр $\theta \le \frac{1}{5} * (\tau - 1)$, где $\tau \ge 1$ - коэффициент возможного увеличения времени выполнения сжатой программы по сравнению со временем выполнения исходной

2. выбрать программы для сжатия, суммарный объем которых больше $\frac{100}{1-\lambda(\theta)}$ Кб, где $\lambda(\theta)$ - это средний коэффициент сжатия (см. рис. 3) системы при заданном θ.

Если условия двух выше описанных пунктов одновременно не могут выполняться или надо получить более высокий коэффициент сжатия программы, то необходимо наложить на БВС требование об увеличении производительности процессора в $\rho = \frac{1+5*\theta}{\tau}$ раз. Например, при $0.2 < \theta < 0.4$ и $\tau = 1.2$ необходимо увеличить производительность процессора в $1.5 < \rho < 1.8$, при этом будет достигнут коэффициент сжатия равный 0.85. А без увеличения производительности процессора максимальный коэффициент сжатия был бы равен 0.95.

Заключение

В данной работе описан метод компактного представления программ на основе частотных характеристик их поведения. На языке C++ создана система компактного представления программ, реализующая исследуемый метод, и проведены её испытания с целью определения зависимости коэффициента сжатия метода от входного параметра системы. Тестовые программы реализуют функциональные задачи, решаемые в БВС современных самолетов. Испытания экспериментально подтвердили возможность применения исследуемого метода в БВС.

Важным отличием данного метода от других методов КПП, является возможность управления степенью сжатия фрагментов программ в зависимости от требуемых условий применения в БВС. В работе даны математические зависимости, позволяющие в заданных условиях получить рациональное решение.

Следует отметить, что данный метод универсален, и может быть применен не только к программам в БВС.

Литература

1. M. Kozuch, A.Wolfe. Compression of embedded system programs. 2000 IEEE International Conference on Computer Design: VLSI in Computers & Processors [HTML, PDF] (http://www.computer.org/portal/site/csdl/)

2. Embedded Computing Design [HTML] (www.embedded-computing.com)

3. К. Колпаков. История развития бортовых цифровых вычислительных машин в России // PCWeek, N32, 1999

4. Лаборатория вычислительных комплексов ВМК МГУ [HTML] (http://lvk.cs.msu.su)

5. Шалимов А.В. Дипломная работа на тему "Исследование метода компактного представления программ на основе частотных характеристик их поведения", МГУ 2007 [PDF] (http://lvk.cs.msu.su/~ashalimov/doc/diplom_ashalimov.pdf)

6. Смелянский Р.Л., Гурьев Д.Е., Бахмуров А.Г. Об одной математической модели для расчета динамических характеристик программы. Программирование, N6, 1986

7. R.L. Smelianski, T. Alanko. On the calculation of control transition probabilities in a program Inform.Processing Letters N.3, 1986

8. P. Brown. Macros without tears // Software: Practice and Experience. Volume 9, Issue 6, Pages 433 - 437, 1979

9. НИР DrTesy [HTML] (http://lvk.cs.msu.su/index.php/articles/65)

10. Смелянский Р.Л., Шалимов А.В., Метод оценки частот выполнения фрагментов кода последовательной программы. // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006

11. Гмурман В.Е. Теория вероятности и математической статистики - 9е издание. М.:Наука, 2003., 363с.

12. Скрипкин В.А., Моисеенко Е.А. Математические методы исследования операций в военном деле. М.:Военное издательство министерства обороны СССР, 1979.

Сутырин П.Г., Мальковский М.Г.

Иерархические конечные автоматы и звездная высота регулярных выражений

Введение

Регулярные языки и описывающие их конечные автоматы и регулярные выражения (см., например, [1]) относятся к наиболее изученным и широко применяемым инструментам теории формальных языков. Одним из параметров сложности регулярного языка является его звездная высота, которая определяется вложенностью звездочек Клини в задающих его регулярных выражениях. Было показано ([2]), что в алфавите $\{a,b\}$ существуют языки любой наперёд заданной звездной высоты. Вопрос об алгоритмической разрешимости задачи вычисления звездной высоты заданного регулярного языка долгое время оставался открытым. В 1989 г. получено решение ([3]), опирающееся на алгебраические свойства регулярных языков, а также на специального вида метрики в графе конечного автомата.

В настоящей работе вводится понятие недетерминированного иерархического конечного автомата, позволяющее естественным образом представить вложенную структуру сильно связных компонент в графе конечного автомата и одновременно звездную высоту эквивалентного регулярного выражения. Описывается алгоритм преобразования конечного автомата в регулярное выражение, включающий в себя построение иерархического конечного автомата по недетерминированному конечному автомату и осуществляющий некоторую оптимизацию сложности иерархии. Это позволяет в ряде случаев получать регулярные выражения меньшей звездной высоты.

1. Определения

Пусть Σ — алфавит, не содержащий символа ε. Пусть $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.

Графом над алфавитом Σ будем называть пару

$$G_\Sigma = (V,E),$$

где V — конечное непустое множество *вершин*, $E \subseteq \{(u,a,v) \mid u,v \in V, a \in \Sigma_\varepsilon\}$ — множество ориентированных помеченных *дуг*. Множество вершин графа G будем обозначать $V(G)$, множество дуг — $E(G)$. Граф G'_Σ будем называть *подграфом* G_Σ, если

$$V(G') \subseteq V,\ E(G') = \{(u,a,v) \in E \mid u,v \in V(G')\}$$

Подграф G' графа G, построенный на подмножестве вершин $V' \subseteq V$ будем обозначать $G' = SubGraph(G, V')$.

Для дуги $e = (u,a,v) \in E$ введем обозначения для *начальной вершины* – $beg(e) = u$, *конечной вершины* – $end(e) = v$, *пометки* – $label(e) = a$.

Путь в графе G — это последовательность вершин и дуг вида $T = v_0e_1v_1e_2v_2...e_mv_m$, такая, что $v_0 \in V$ и для всех $i = 1,...,m$ $v_i \in V$ и $e_i = (v_{i-1},a_i,v_i) \in E$. Будем говорить, что T — это путь *из* v_0 *в* v_m и обозначать его как $T(v_0,v_m)$. Для $m = 0$ путь $T = v_0$ будем называть *пустым*.

Если существует путь $T(u,v)$, то будем говорить, что вершина v *достижима* из вершины u, или что вершина u *достигает* вершину v.

Непустой путь T будем называть *циклом*, если $beg(T) = end(T)$. Цикл T будем называть *простым*, если он не содержит фрагментов, являющихся циклами.

Множество всех путей в G будем обозначать $Paths(G)$. Множество всех путей в G из вершины u в вершину v будем обозначать $Paths(G,u,v)$.

Пометку $label(T) \in \Sigma^*$ пути T определим рекурсивно:

- если T — пустой путь, положим $label(T) = \lambda$;
- если T — путь из одной дуги e и $label(e) = \varepsilon$, то $label(T) = \lambda$;
- если T — путь из одной дуги e и $label(e) = a \in \Sigma$, то $label(T) = a$;
- для непустого пути $T = eT'$, где e — дуга, положим $label(T) = label(e)label(T')$.

Два графа G_1 и G_2 будем называть *изоморфными*, если существует биективное отображение f: $V(G_1) \rightarrow V(G_2)$, такое что $E(G_2) = \{(f(u),a,f(v)) \mid (u,a,v) \in E(G_1)\}$ В дальнейшем любые два изоморфных графа будем считать равными.

Конечный автомат анализирует входную цепочку, читая её слева направо. Если после этого он оказывается в заключительном состоянии, то говорят, что автомат допускает данную цепочку. Мы обратимся к эквивалентной графической форме конечного автомата, обычно называемой *диаграммой переходов* ([6]).

Итак, состояния автомата представляются вершинами графа, функция переходов определяет расположение помеченных дуг, а успешное вычисление соответствует пути из некоторой начальной вершины в некоторую заключительную. Допускаемая вычислением цепочка есть пометка этого пути. Таким образом, язык, допускаемый автоматом, определяется как множество пометок подмножества путей в графе.

Недетерминированным конечным автоматом (НКА) над алфавитом Σ будем называть тройку

$$A_\Sigma = (G_\Sigma, V_s, V_e),$$

где $G_\Sigma = (V,E)$ — граф над алфавитом Σ; $V_s, V_e \in V$ — подмножества выделенных *начальных* и *заключительных* вершин соответственно. Начальные и заключительные вершины вместе будем называть *терминальными*. Индекс Σ будем опускать, если алфавит подразумевается. Недетерминизм автомата выражен в том, что для фиксированных $u \in V$ и $a \in \Sigma$ может существовать несколько дуг вида $(u,a,v) \in \Sigma$. Введем обозначения для элементов автомата: $G(A_\Sigma) = G_\Sigma$; $V_s(G) = V_s$; $V_e(G) = V_e$.

Предложением конечного автомата A будем называть путь T в графе G_Σ, такой что $beg(T) \in V_s$ и $end(T) \in V_e$. Множество всех предложений автомата A обозначим через $Sentences(A)$. *Линией* будем называть предложение автомата A, являющееся простым путем. Множество всех линий автомата A обозначим через $Lines(A)$.

Язык, допускаемый конечным автоматом A_Σ, обозначим $L(A_\Sigma)$ и определим следующим образом:

$L(A_\Sigma) = \{label(T) \mid T \in Sentences(A_\Sigma)\} = label(Sentences(A_\Sigma))$, $L(A_\Sigma) \subseteq \Sigma^*$.

Определим автомат $Empty = (\{\{v_1,v_2\},\varnothing),\{v_1\},\{v_2\})$, допускающий язык $L(Empty) = \varnothing$.

Два НКА A_1 и A_2 будем называть *изоморфными*, если изоморфны $G(A_1)$ и $G(A_2)$ с отображением f, причем $f(V_s(A_1)) = V_s(A_2)$ и $f(V_e(A_1)) = V_e(A_2)$. В дальнейшем любую пару изоморфных НКА A_1 и A_2, будем считать равными: $A_1 = A_2$.

Приведенным будем называть конечный автомат $A = (G,V_b,V_e)$, обладающий следующими свойствами:

- $|V_b| = 1$ и $|V_e| = 1$;
- если $L(A) = \varnothing$, то $A = Empty$;
- если $L(A) \neq \varnothing$, то каждая вершина $G(A)$ входит хотя бы в одно предложение A.

В записи приведенного автомата $A = (G,\{v_b\},\{v_e\})$ в дальнейшем будем опускать фигурные скобки при указании входной и выходной вершин: $A = (G,v_b,v_e)$. Легко конструктивно доказывается следующая

Лемма 1.1. *Для каждого конечного автомата A существует эквивалентный ему приведенный конечный автомат A'.*

Обозначим класс всех приведенных автоматов через NFA.

Пусть $A = (G,V_b,V_e)$ — конечный автомат. Вершина v автомата A является *точкой сочленения* автомата A, если выполнены следующие условия:

- $v \notin V_b \cap V_e$;
- v входит в каждое предложение A;
- v не входит ни в один цикл $G(A)$.

Граф G будем называть *сильно связным*, если для всякой пары вершин $u,v \in V(G)$ одновременно u достигает v и v достигает u.

Пусть B — подграф G. Будем называть B *секцией (сильно связной компонентой)* графа G, если B является сильно связным и либо $B = G$, либо всякий подграф B' графа G, такой что $V(B') \supset V(B)$, не является сильно связным. Секцию $B = (\{v\},\varnothing)$ графа G, где $v \in V(G)$, будем называть *тривиальным*. Очевидно, что каждая вершина графа принадлежит ровно одной секции (возможно, тривиальной). Множество всех секций графа G обозначим через $Sections(G)$.

Пусть $A = (G, v_b, v_e) \in NFA$ — НКА, B — секция графа G, и $v \in V(B)$. Вершину $v \in V(B)$ будем называть *входом (входной вершиной)* секции B в двух случаях:

- если $v = v_b$,
- если найдутся вершина $u \in V(G)\text{-}V(B)$ и дуга $(u,a,v) \in E(G)\text{-}E(B)$.

Вершину $v \in V(B)$ будем называть *выходом (выходной вершиной)* секции B в двух случаях:

- если $v = v_e$,

если найдутся вершина $w \in V(G)\text{-}V(B)$ и дуга $(v,a,w) \in E(G)\text{-}E(B)$.

Множества входов и выходов секции B обозначим соответственно $In(B)$ и $Out(B)$. Если $|In(B)| = m$, $|Out(B)| = n$, то B будем называть (m,n)-*секцией*. Заметим, что $m,n \geq 1$, поскольку отсутствие входа делает вершины секции недостижимыми из начальной вершины, а отсутствие выхода таким же образом «отделяет» секцию от заключительной вершины автомата. Обозначим через $In(B,i)$ и $Out(B,j)$ соответственно i-й вход и j-й выход секции B, $1 \leq i \leq m$, $1 \leq j \leq n$.

Разобьем множество всех дуг $\{(u,a,v) \mid u,v \in V(G), a \in \Sigma\}$ автомата $A = (G_\Sigma, v_b, v_e)$ на два класса: *внутренние* — если u и v принадлежат одной и той же секции (в частности, если $u = v$) и *внешние* — если u и v принадлежат разным секциям. Таким образом, внешняя дуга всегда соединяет выход некоторой секции с входом другой секции.

Определим функцию $Exclude(G,e)$ для графа G и его дуги e следующим образом:

$$Exclude(G, e) = Trim\ ((V(G), E(G)\text{-}\{e\}))$$

Определим теперь для графа ключевое понятие *ранга*. Оно отражает максимальную глубину вложенности сильно-связных компонент. Вложенными в данную СС-компоненту считаются все СС-компоненты, которые образуются после удаления некоторой дуги. Эта дуга в определенном смысле «отвечает» за сильную связность текущей компоненты. Допустимым является и такой случай, когда при удалении любой дуги сильная связность компоненты сохраняется.

Определим рекурсивно *ранг* графа G как целочисленную функцию $rank(G)$:

- если G — тривиальная секция, то $rank(G) = 0$;

• если G — нетривиальная секция, то

$rank(G) = 1 + min\{rank(Exclude(G,e)) \mid e \in E(G)\}$, при этом все дуги, на которых достигается этот минимум, будем называть *ранговыми* для нетривиальной секции G;

• в остальных случаях

$rank(G) = max\{rank(B) \mid B$ является секцией графа $G\}$.

Отметим важное свойство ранговых дуг: если $rank(G) \geq 1$ и e — ранговая дуга G, то $rank(Exclude(G,e)) = rank(G) - 1$.

Пусть Σ — алфавит, не содержащий символов $^*,+,\varepsilon,\varnothing,(,)$. Согласно [5] и [3] определим рекурсивно *регулярное выражение* γ над Σ, задаваемый им *язык* $L(\gamma)$, и его *звездную высоту* $sh(\gamma)$:

- $\gamma = a$, где $a \in \Sigma, \gamma = \varepsilon$ и $\gamma = \varnothing$ — регулярные выражения; $L(a) = \{a\}, L(\varepsilon) = \{\Lambda\}, L(\varnothing) = \varnothing$; во всех случаях $sh(\gamma) = 0$.
- Если α и β — регулярные выражения, то:
- $\gamma = \alpha + \beta$ — регулярное выражение (*сумма*), и $L(\alpha + \beta) = L(\alpha) \cup L(\beta)$; $sh(\gamma) = max\{sh(\alpha), sh(\beta)\}$.
- $\gamma = (\alpha)(\beta)$ — регулярное выражение (*конкатенация*), и $L((\alpha)(\beta)) = L(\alpha)L(\beta)$; в случае, если α и β не являются суммами, скобки можно опустить; $sh(\gamma) = max\{sh(\alpha), sh(\beta)\}$.
- $\gamma = (\alpha)^*$ — регулярное выражение (*итерация*), и $L((\alpha)^*) = L(\alpha)^*$; $sh(\gamma) = sh(\alpha) + 1$.
- Других регулярных выражений над Σ нет.

Регулярное выражение α и конечный автомат A будем называть эквивалентными, если $L(\alpha) = L(A)$, и обозначать: $\alpha \sim A$.

Класс всех регулярных выражений над алфавитом Σ обозначим RE_Σ.

Иерархический конечный автомат естественным образом расширяет понятие обычного автомата. С каждой вершиной можно связать «вложенный» автомат. Предложения иерархического автомата конструируются из предложений автомата верхнего уровня при помощи подстановки вместо всех вершин некоторых предложений вложенных автоматов.

Чтобы избежать бесконечных подстановок (и выхода из класса регулярных языков), будем рассматривать автоматы лишь с конечной глубиной этой вложенности.

Итак, рекурсивно определим иерархический конечный автомат (ИКА) H и его *глубину* $depth(H)$:

• символ *nil* — это ИКА, называемый *атомарным* (тривиальным); $depth(nil) = 0$;

• пусть $A = (G, v_b, v_e) \in NFA$ — конечный автомат с $n \geq 1$ вершинами: $V(G) = \{v_1, \ldots, v_n\}$, пусть $H_1, \ldots, H_n$ — набор из n иерархических конечных автоматов, и пусть

subaut = $\{(v_i, H_i) \mid i = 1, \ldots, n\}$; тогда четверка $H = (G, v_b, v_e, subaut)$ — это ИКА, и $depth(H) = 1 + max_{1 \le i \le n}(depth(H_i))$;

- Других ИКА нет.

Заметим, что четвертый элемент нетривиального ИКА — *subaut* — можно трактовать как функцию, заданную на $V(G)$ и писать $H_i = subaut(v_i)$, если $(v_i, H_i) \in subaut$.

В нетривиальном иерархическом автомате $H = (G, v_b, v_e, subaut)$ будем называть v_b *начальной вершиной*, v_e — *заключительной* вершиной, *subaut* — *функцией подчинения*. Введем для элементов автомата H обозначения: $G(H) = G$; $v_b(H) = v_b$; $v_e(H) = v_e$, $subaut_H = subaut$.

Вершину v иерархического автомата H будем называть *простой*, если $subaut(v) = nil$, иначе — *автоматной*. Множество всех автоматных вершин обозначим $V_{aut}(H)$. Автоматы $subaut(v)$ для $v \in V_{aut}(H)$ будем называть *подавтоматами (подчиненными автоматами)* автомата H.

Класс всех иерархических конечных автоматов обозначим *HFA (hierarchial finite automata)*.

Иерархические автоматы глубины 1 будем называть *плоскими*. Если задан конечный автомат $A = (G, v_b, v_e)$, то через $plain(A)$ обозначим следующий плоский ИКА:

$plain(A) = (G, v_b, v_e, subaut)$, где $subaut = \{(v, nil) \mid v \in V(G)\}$.

Объединенное множество дуг иерархического автомата H, обозначаемое $E^\circ(H)$, определим следующим образом. Для $H = nil$ положим $E^\circ(H) = \varnothing$, для плоского H положим $E^\circ(H) = E(H)$, а для всех остальных автоматов H определим

$$E^\circ(H) = E(H) \cup \bigcup_{v \in Vaut(H)} E^\circ(subaut(v))$$

Можно предполагать, что все объединяемые множества не пересекаются[1]. Пометка цепочки T над алфавитом $E(H)$ определяется по аналогии: $label(\lambda) = \lambda$; $label(eT) = label(e)label(T)$.

Далее рекурсивно определим предложение иерархического автомата как цепочку $T \in E(H)^*$, получаемую согласно иерархии подавтоматов.

Пусть $H = (G, v_b, v_e, subaut)$ — ИКА. Определим *предложение* T автомата H, $T \in (E(H))^*$ и множество $Sentences(H)$ всех предложений автомата H, $Sentences(H) \subseteq (E(H))^*$.

Если $depth(H) = 0$, т.е. $H = nil$, то $T = \lambda$ — предложение H. В этом случае $Sentences(H) = \{\lambda\}$.

Если $depth(H) = 1$, т.е. H — плоский, то предложениями ИКА H будем считать все предложения конечного автомата (G, v_b, v_e). Таким образом, $Sentences(H) = Sentences((G, v_b, v_e))$.

---

[1] Во избежание совпадений все вершины в составе ребер можно систематически переименовать.

Если же $depth(H) \geq 2$, то для всякого пути $T \in Paths(G, v_b, v_e)$, $T = v_0 e_1 v_1 \ldots e_m v_m$, $m \geq 0$ определим множество

$Expand\ (T) = \{T0e1T1e2T2\ \ldots e,mTm \mid Ti \square Sentences(subaut(vi)), i = 0,\ldots,k\}$.

Элемент $Expand(T)$ — это цепочка в алфавите $E(H)$, полученная из пути в графе G подстановкой вместо всех вершин некоторых предложений соответствующих подавтоматов.

Предложениями H будем считать все элементы множества $Sentences(H)$, определенного следующим образом:

$$Sentences(H) = \bigcup_{T \in Paths(G, vb, ve)} Expand(T)$$

Корректность этих определений легко доказать индукцией по глубине автомата.

Язык, задаваемый иерархическим автоматом H, обозначим $L(H)$ и определим:

$$L(H) = \{label(T) \mid T \in Sentences(H)\}.$$

Пару автоматов A_1 и A_2 из классов *NFA* или *HFA* будем называть *эквивалентными*, если $L(A_1) = L(A_2)$, и обозначать $A_1 \sim A_2$.

Следующие две леммы устанавливают, что классы *NFA* и *HFA* задают один и тот же класс (регулярных) языков.

Лемма 1.2. *Пусть $A \in NFA$— конечный автомат. По A можно построить такой иерархический конечный автомат $H \in HFA$, что $L(H) = L(A)$.*

Доказательство. В качестве H следует использовать $plain(A)$.

Лемма 1.3. *Пусть $H \in HFA$ — иерархический конечный автомат. По H можно построить такой конечный автомат $A \in NFA$, что $L(A) = L(H)$.*

Доказательство. Конечный автомат A можно получить из иерархического автомата H путем рекурсивной подстановки соответствующих подавтоматов вместо автоматных вершин и добавления недостающих дуг. Подстановка в любом случае завершается, так как глубина ИКА конечна.

Определим теперь ряд подклассов (типов) иерархических автоматов. Далее, введем для автоматов выделенных типов понятие структурной записи, в которой помимо обычных графа, выделенных вершин и функции подчинения будет содержаться информация о типе и, частично, о структуре автомата. В определениях типов ИКА мы будем накладывать ограничения только на граф автомата, не затрагивая структуры подавтоматов.

Итак, *структурной записью (СЗ)* иерархического автомата H будем называть выражение $S(H)$, определенное следующим образом. $S(nil) = nil$ — структурная запись автомата *nil*.

Именованный кортеж вида

$$S(H) = name <elem_1, elem_2, ..., elem_n>,\ n \geq 1$$

— структурная запись автомата H, где *name* — это тип записи (символическое имя), а каждый $elem_i$ — это один из следующих объектов:

- пометка дуги ($elem_i = a \in \Sigma_\varepsilon$);
- множество из k пометок дуг ($elem_i = \{a_i\}_{i=1,...,k} \subseteq \Sigma_\varepsilon$);
- структурная запись иерархического КА;
- конечное множество структурных записей иерархических КА.

Для СЗ $S = name<elem_1, ..., elem_n>$ обозначим $Type(S) = name$, $Elements(S) = \{elem_i\}_{i=1,...,n}$

Отметим, что в данном определении не фиксируется, как связаны между собой части $elem_i$, поэтому возможно неоднозначное восстановление иерархического автомата по записи. Поэтому, определяя далее подклассы иерархических автоматов, будем обеспечивать однозначность соответствия структурной записи иерархическому автомату.

Класс всех структурных записей обозначим *SR*.

Для класса записей $S^\circ \subseteq SR$, введем множество $Restrict(S^\circ)$, состоящее из всех записей из S°, в которые на всех уровнях вложенности входят лишь записи из $S^\circ \cup \{nil\}$.

Класс *Raw* («неструктурированный») совпадает с *HFA*.

Структурную запись для $H \in Raw$ определим следующим образом:

$$S(H) = raw<H, \{S_i\}_{i=1,...,V_{aut}(H)}>,$$

где $\{S_i\} = S(V_{aut}(H))$. В данном случае дополнительной информации об автомате в записи нет, и он входит в запись в явном виде, вместе со всеми нетривиальными подавтоматами. Такой класс нам потребуется для представления еще не проанализированного автомата.

Класс *Atomic* («атомарный») состоит из одного элемента *nil*:

$$Atomic = \{nil\},\ S(nil) = nil.$$

Класс *Loopfree* («без циклов») состоит из всех иерархических автоматов $H = (G, v_b, v_e, subaut)$, таких что G не содержит циклов. Автомат $H \in Loopfree$ будем структурно записывать следующим образом:

$$S(H) = loopfree<H, \{S_i\}_{i=1,...,V_{aut}(H)}>,$$

где $\{S_i\} = \{S(subaut(v)) \mid v \in V_{aut}(H)\}$.

Этот класс мы будем использовать для представления автоматов без циклов.

Класс *Section* («секция») состоит из всех иерархических автоматов таких что $G = G_{section}(a)$, где $G_{section}(a)$ — «эталонный» для данного подкласса граф, параметризованный пометкой $a \in \Sigma_\varepsilon$.

$$V(G_{section}) = \{v_b, v_e, v_0, v_m, v_{cycle}, v_{out}, v_{pass}\},$$

$E(G_{section}(a))$ = $\{(v_b,\varepsilon,v_{pass}),(v_{pass},\varepsilon,v_e),(v_b,\varepsilon,v_{in}),(v_{in},a,v_0),(v_0,\varepsilon,v_{cycle}),(v_{cycle},a,v_0),(v_0,\varepsilon,v_{out}),(v_{out},\varepsilon,v_e)\}$

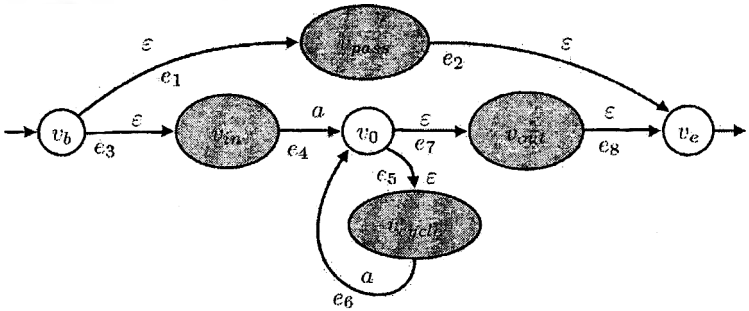


Рис. 1: **Граф $G_{section}(a)$**

Граф $G_{section}$ изображён на рисунке 1: у каждой дуги указана ее пометка и обозначение (снизу), начальная вершина отмечена входящей стрелкой, заключительная — исходящей.

Автоматных вершин четыре: $V_{aut}(H) = \{v_{in}, v_{cycle}, v_{out}, v_{pass}\}$, так что иерархические автоматы класса *Section* отличаются друг от друга пометкой a и подавтоматами.

Структурную запись для автомата $H \in Section$ определим так:

$S(H) = section\langle S_{in}, S_{cycle}, S_{oub}, S_{pass}, a\rangle$,

где

$S_{In} = S(subaut(v_{In}))$, $S_{out} = S(subaut(v_{out}))$,

$S_{pass} = S(subaut(v_{pass}))$, $S_{cycle} = S(subaut(v_{cycle}))$

— структурные записи подавтоматов H; a — единственная непустая пометка дуг H.

Способ однозначного построения иерархического автомата класса *Section* по СЗ указанного вида полностью определяется графом $G_{section}(a)$.

Этот класс будет использоваться для представления сильно связных автоматов, «разложенных» относительно ранговой дуги.

Обратными дугами иерархического автомата H будем называть все дуги вида $(end(H), a, beg(H))$, множество всех обратных дуг обозначим $Backs(H)$.

Класс *Backsum* («сумма обратных дуг») состоит из всех иерархических автоматов $H = (G, v_{core}, v_{core}, subaut)$, таких, что $G = G_{backsum}$, где $G_{backsum}(\{a_1,...,a_k\})$ — эталонный для класса *Backsum* граф, параметризованный множеством пометок $\{a_1,...,a_k\} \in \Sigma, k \geq 1$. Он определяется следующим образом:

$$V(G_{backsum}) = \{v_{core}\}, E(G_{backsum}) = \{(v_{core}, a_i, v_{core}) \mid i = 1,...,k\}.$$

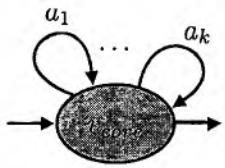


Рис. 2: Граф $G_{backsum}(\{a_1,\ldots,a_k\})$

Граф $G_{backsum}(\{a_1,\ldots,a_k\})$ изображён на рисунке 2. Структурную запись автомата *H Backsum* определим следующим образом:

$$S(H) = backsum <S_{core}, \{a_i\}_{i=1,\ldots,k}>,$$

где $S_{core} = S(subaut(v_{core}))$, a_i — пометки всех дуг.

Этот класс будет использоваться для представления автоматов с «вынесенными» обратными дугами.

Определим теперь множества структурных записей

S_{basic} = $S(Atomic) \cup S(Loopfree) \cup S(Section) \cup S(Backsum)$,

$S_{canonical}$ = $Restrict(S_{basic})$,

$S_{canonical+raw}$ = $Restrict(S_{basic} \cup S(Raw))$.

Для записи $S \in S_{canonical+raw}$ будем обозначать через $H(S)$ иерархический автомат, который ей однозначно соответствует. Для S определим также $beg(S) = beg(H(S))$, $end(S) = end(H(S))$ $G(S) = G(H(S))$ $V_{all}(S) = V_{all}(H(S))$, $subaut_S = subaut_{H(S)}$.

Каноническими будем называть все автоматы $H(S)$ для $S \in S_{canonical}$. Обозначим их класс *CHFA (*canonical HFA).

Лемма 1.4. *Существует взаимно однозначное соответствие между $S_{canonical}$ и CHFA.*

Доказательство. Каждой записи $S \in S_{canonical}$ по определению однозначно соответствует автомат $H(S)$. Классы *Atomic*, *Loopfree*, *Section*, и *Backsum* попарно не пересекаются, так как входящие в них автоматы всегда различаются наличием циклов и числом вершин. Каждый канонический автомат принадлежит ровно одному из этих классов, поэтому имеет единственную структурную запись соответствующего типа.

Лемма 1.4 позволит нам в дальнейшем отождествлять канонические иерархические автоматы и их структурные записи.

Определим теперь для канонического ИКА S величину *cycledepth*(S), называемую *циклической глубиной*:

- Если $S = nil$, то $cycledepth(S) = 0$;
- Если $S = loopfree<S, \{S_i\}_{i=1,\ldots,k}>$, то $cycledepth(S) = max_{1 \le i \le k}\{cycledepth(Si)\}$;
- Если $S = section<S_{in}, S_{cycle}, S_{out}, S_{pass}, a>$, то $cyclepth(S) = 1 + cycledepth(S_{cycle})$
- Если $S = backsum<S_{core}, \{a_i\}_{1,\ldots,k}>$, то

cyclepth(S) =1 + cycledepth(S_{core})

2. Алгоритм

Определим алгоритм α построения регулярного выражения по конечному автомату. также будем рассматривать как функцию

$$\alpha : NFA \rightarrow RE$$

и записывать: $E = \alpha(A)$ для $A \in NFA, E \in RE$.

По аналогии с [8] опишем алгоритм на псевдокоде, основанном на языке программирования Паскаль.

Выражение `|X|` обозначает мощность множества X, оператор `return y;` используется для завершения работы функции с возвратом значения *y*. Все выражения вида `beg(G)`, `plain(A)` используются согласно определениям.

На псевдокоде алгоритму α будет соответствовать функция `fa_to_re`:

```
function fa_to_re( A )
begin
  S := fa_to_chfa( A );
  E := chfa_to_re( S );
  return E;
end;
```

где `A` — исходный конечный автомат, `E` — получаемое выражение.

Функция `fa_to_chfa(A)` преобразует конечный автомат в эквивалентный ему канонический иерархический конечный автомат.

Фукция `chfa_to_re(S)` преобразует канонический иерархический конечный автомат в эквивалентное ему регулярное выражение.

Определим функцию `fa_to_chfa(A)`. Она вызывает вспомогательные функции следующих видов.

Группа функций с префиксом `find_` отвечает за анализ графов КИКА на текущем уровне рекурсии. Каждая из функций с префиксом `build_` строит новый КИКА на основе текущего КИКА и результатов анализа на текущем уровне. Каждая из функций с префиксом `process_` отвечает за полную (рекурсивную) обработку построенного КИКА определенного типа.

```
function fa_to_chfa( A )
begin
  return process_raw( build_raw( A ) );
end;

function build_raw( A )
begin
```

```
  return raw< plain( Trim( A ) ), V_aut( A ) >
end;
```

Конечный автомат преобразуется в плоский КИКА типа *raw* и подается на вход процедуре process_raw.

```
function process_raw( S )
begin
  if is_atomic( S ) then
  begin
    return nil;
  end;
  sections := find_sections( S ); // |sections| > 0
  if |sections| = 1 then
  begin
    backs := find_backs( S );
    if |backs| = 0 then
    begin
      rank_edge := find_rank_edge( S );
      return process_section( build_section( S, rank_edge ) );
    end;
    else begin
      return process_backsum( build_backsum( S, backs ) );
    end;
  end;
  else begin // |sections| > 1
    return process_loopfree( build_loopfree( S, sections ) );
  end;
end;

function is_atomic( S )
begin
  return ( |V(S)| = 1 and |E(S)| = 0);
end;

function find_backs( S )
begin
  return { e in E(G(S)) | e = (end(S), a, beg(S)) }
end;

function find_sections( S )
begin
  // ... работает процедура поиска СС-компонент в графе G(S)
  // ... и возвращается их множество
end;

function find_rank_edge( S )
begin
  // ... вычисляется по определению ранг G(S)
  // ... и возвращается любая из ранговых дуг
end;
```

Вначале производится проверка атомарности S. Затем — поиск сильно связных компонент в графе $G(S)$ (find_sections). Если граф оказался сильно связным, то производится поиск обратных дуг (find_backs). Если обратных дуг нет, то в графе выделяется ранговая

дуга (find_rank_edge), строится и обрабатывается КИКА типа *section*, иначе строится и обрабатывается КИКА типа *backsum*.

Если сильно связных компонент оказалось несколько, то строится и обрабатывается КИКА типа *loopfree*.

Функция find_sections реализует известный ([7],[8],[9]) алгоритм поиска сильно связных компонент, а функция find_rank_edge ищет ранговую дугу согласно определению.

Рассмотрим построение КИКА типа *section*.

```
function build_section( S, rank_edge )
begin
  G1 := Exclude( G(S), rank_edge );
  v_b := beg( H );
  v_e := end( H );
  v1 := beg( rank_edge );
  v2 := end( rank_edge );

  in := build_raw( (G1, v_b,, v_1 );
  cycle := build_raw( (G1, v_2, v_1) );
  out := build_raw( (G1, v_2, v_e) );
  pass := build_raw( (G1, v_s, v_e}) );

  return section< in, cycle, out, pass, label(rank_edge) >;
end;
```

Вначале из графа $G(S)$ исключается ранговая дуга, затем строятся четыре КИКА типа *raw* с определенными парами начальных и заключительных вершин, и, наконец, конструируется КИКА типа *section*.

Рассмотрим построение КИКА типа *backsum*.

```
function build_backsum( S, backs )
begin
  G1 := Exclude( G(S), backs );
  return backsum< build_raw( (G1, beg(S), end(S)), subaut(S) ),
                  label(backs) >;
end;
```

Из графа $G(H)$ исключаются обратные дуги[1], после чего строится КИКА типа *backsum*, имеющий в качестве S_{core} КИКА типа *raw*.

Рассмотрим построение КИКА типа *loopfree*.

```
function build_loopfree( S, sections )
begin
  S1 := build_empty()
  for B in sections
  begin
    // ... добавить группу вершин в S1
   end;
  for e in find_external_edges( H )
  begin
    // ... добавить группу дуг в S1
  end;
```

[2] Используется вариант полиморфной функции *Exclude*, исключающий несколько дуг.

```
  // ... склеить терминальные вершины
  return S1;
end;

function build_empty()
begin
  return ( ( {}, {} ), {}, {} )
end;

function find_external_edges( H )
begin
  // ... вычисляет и возвращает множество внешних дуг H
end;
```

Для каждой секции B из множества *sections* строится группа автоматных вершин в результирующем КИКА $S1$. Пусть $B \in sections$ — (m,n)-секция, $m,n \geq 1$. Тогда строится группа вершин

$$\{v^B_{i,j} \mid 1 \leq i \leq m,\ 1 \leq j \leq n\},$$

т.е. по одной вершине для каждой пары "вход-выход" секции B. Каждой вершине $v_{i,j}^B$ сопоставляется плоский подавтомат следующего вида:

$$subaut(v_{i,j}^B) = plain\ ((B, In\ (B,\ i), Out(B,j))$$

Дуги в КИКА добавляются по следующему правилу: если в $G(H)$ есть внешняя дуга (u,a,v), то в КИКА необходимо добавить все дуги вида $(Out(B_1,j),a,In(B_2,i))$, где $Out(B_1,j) = u$ и $In(B_2,i) = v$.

Далее выделяются подмножества вершин

$$V_s = \{v_{i,j}^B \in V(H1) \mid \exists\ B,\ i:\ beg(H) = In(B,i)\}$$

$$V_e = \{v_{i,j}^B \in V(H1) \mid \exists B, j:\ end(H) = Out(B,j)\},$$

и в H добавляются две простые вершины v_s и v_e, а также группы дуг $\{(v_s,\varepsilon,v) \mid v \in V_s\}$ и $\{(v,\varepsilon,v_e) \mid v \in V_e\}$.

Вершины v_s и v_e полагаются начальной и заключительной в $H1$.

Вспомогательная функция `dive` отвечает за косвенную рекурсию. Она ищет в КИКА H все (возможно, косвенно подчиненные) подавтоматы H' типа *raw* и заменяет их на результат вычисления `process_raw(H')`.

Далее:

```
function process_backsum( S )
begin
  return dive( S )
end;

function process_section( S )
begin
  return dive( S );
end;
```

Обработка КИКА типов *backsum* и *section* сводится к вызову *dive*.

```
function process_loopfree( S )
begin
  junctions := find_junctions( S );
```

```
  if |junctions| = 0 then
  begin
    return dive( S );
  end;
  else begin
    return dive( split_junctions( S, junctions ) );
  end;
end;
```

Обработка КИКА типа *loopfree* заключается в поиске последовательности $v_1,...,v_k$ точек сочленения (`find_junctions`), таких что граф $G(H)$ имеет вид, представленный на рисунке 3, и построении с помощью функции `split_junctions` эквивалентного КИКА типа *loopfree*, представленного на рисунке 4.

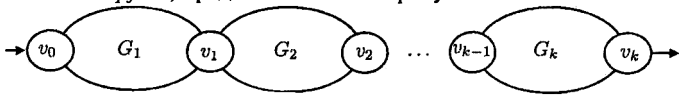


Рис. 3. **Точки**

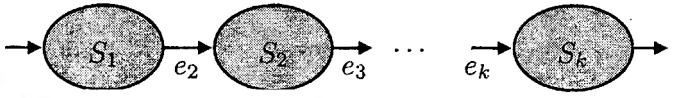


Рис. 4. **Эквивалентный канонический**

A_i = `build_raw`$((G_i, v_{i-1}, v_i)), i = 1,...,k$.

Функция `chfa_to_re(H)` строит по каноническому ИКА регулярное выражение согласно выражениям (1)–(4):

$$E(nil) = \lambda \quad (1)$$

$$E(loopfree\langle S, \{S_i\}\rangle) = \sum_{T \in Lines(S)} E'(T), \quad (2)$$

где для линии $T = v_0 e_1 v_1 ... e_m v_m$, $m \geq 0$

$E'(T) = E(sub(v_0))label(e_1)E(sub(v_1))...label(e_m)E(sub(v_m))$

Для автомата $S = section\langle S_{in}, S_{cycle}, S_{out}, S_{pass}, a\rangle$

$$E(S) = E(S_{pass}) + E(S_{in})a(E(S_{cycle})a)^* E(S_{out}), \quad (3)$$

и, наконец

$$E(backsum\langle S_{core}, \{a_i\}_1^k\rangle) = E(S_{core})((a_1 + ... + a_k)E(S_{core}))^* \quad (4)$$

3. Обоснование алгоритма

Приведём схемы доказательств завершаемости и корректности алгоритма.

Введем *размер* НКА A как целочисленную функцию $size(A)$:

$$size(A) = |E(A)|$$

Введем аналогичное понятие *размера* КИКА S:

$$size(nil) = 0,\ size(S) = |E(S)| + \sum_{v \in Vaut(S)} size(subaut(v)).$$

Очевидно, что $size(plain(A)) = size(A)$.

Доказательство завершаемости и корректности алгоритма можно провести по индукции по размеру КИКА. Нужно показать, что на всех рекурсивных ветвях вычисления размер автоматов уменьшается (см. рисунок 5).

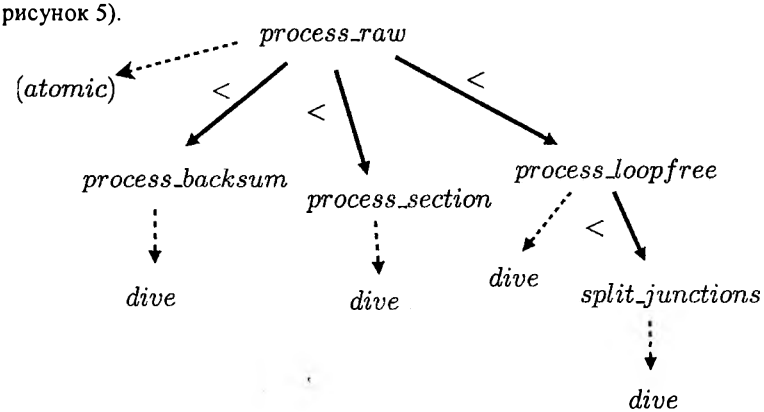


Рис. 5: Схема рекурсии в `process_raw`

Лемма 3.1. *Для каждого конечного автомата $A \in NFA$ вызов* `fa_to_chfa(A)` *завершается.*

Доказательство. Будем предполагать, что вызовы всех не рекурсивных функций с префиксами `find_` и `build_` завершаются.

Базис индукции. Пусть $size(A) = 0$. Тогда полученный по нему плоский S — тривиальный, и вызов завершается.

Шаг индукции. Пусть $size(A) = size(S) = s > 0$, а для для всех НКА размера меньше s, вызов завершается. Поскольку в этом случае S не тривиальный, строится и обрабатывается рекурсивно КИКА S' одного из типов *Section,Backsum,* или *Loopfree*. Нетрудно показать, что в каждом из этих случаев

$\forall v \in V_{aut}(S')\ size(subaut(v)) < s,$

откуда по предположению индукции следует завершаемость.

Лемма 3.2. *Для каждого КИКА S вызов* `chfa_to_re(S)` *завершается.*

Доказательство. Поскольку автомат S имеет конечную глубину, вычисление регулярного выражения по соотношениям (1)–(4) завершится.

Теорема 3.1. *Для любого конечного автомата $A \in NFA$ вычисление $\alpha(A)$ завершается.*

Доказательство. Утверждение теоремы следует из лемм 3.1 и 3.2.

Установим теперь корректность алгоритма, а именно, что $\alpha(A) = E \sim A$.

Лемма 3.3. *Для любого конечного автомата* $A \in NFA$ *функция* `fa_to_chfa(A)` *строит канонический автомат* $S \in S_{canonical}$: $S \sim A$.

Доказательство. Достаточно показать корректность функции `process_raw`(S_0), что мы сделаем индукцией по размеру $size(S_0)$ канонического автомата $S_0 = plain(A)$ типа *raw*. Очевидно, что $S_0 \sim A$, поэтому убедимся, что $process\_raw(S_0) = S \sim S_0$.

Базис индукции: $size(S_0) = 0$. В этом случае $S_0 = nil$, вычисление идет по нерекурсивной ветви и сразу строится $S = nil$. Утверждение леммы верно.

Шаг индукции: $size(S_0) > 0$, и для всех канонических автоматов S' : $size(S') < size(S_0)$ справедливо $process\_raw(S') \sim S'$.

В зависимости от свойств S_0, строится и рекурсивно обрабатывается канонический автомат S_1 : $Type(S_1) \in \{section, loopfree, backsum\}$. Можно показать, что во всех трех случаях после завершения рекурсивного вычисления будет справедливо $S_1 \sim S_0$.

Лемма 3.4. *Для любого канонического автомата* $S \in S_{canonical}$ *функция* `chfa_to_re(S)` *строит регулярное выражение* $E \in RE$: $E \sim S$, *причем* $sh(E) = cycledepth(S)$.

Доказательство. Применим математическую индукцию по $size(S)$.

Базис индукции. $size(S) = 0$, т.е. $S = nil$, и по (1) $E = \lambda$, $L(E) = \{\lambda\} = L(S)$, $sh(E) = cycledepth(S) = 0$.

Шаг индукции. $size(S) > 0$ и $\forall S' \in S_{canonical}$: $size(S') < size(S)$ утверждение леммы верно. В частности, для подавтоматов: $\forall S' \in Substruct(S)$ $E' = chfa\_to\_re(S') \sim S'$, $sh(E') = cycledepth(S')$ (см. доказательство леммы 3.1).

Возможны случаи: $Type(S) \in \{section, loopfree, backsum\}$, и можно показать, что в каждом из них утверждение леммы верно.

Теорема 3.2. *Для любого конечного автомата* $A \in NFA$ *алгоритм* α *строит регулярное выражение* $E = \alpha(A)$: $E \sim A$.

Доказательство. Утверждение теоремы следует из лемм 3.3 и 3.4.

Заключение

В работе предложен способ выявления в графе конечного автомата иерархических структур, отвечающих за звездную высоту регулярного выражения и задаваемого им языка. Постановка и исследование в таких терминах задач, связанных с регулярными языками, может привести к качественно новым их решениям. Так, известной и достаточно трудоёмкой ([3]) задаче о звездной высоте регулярного языка соответствует задача о минимизации иерархического конечного автомата по глубине, и можно рассчитывать, что решение последней будет более прозрачным.

Литература

1. Yu S., Regular languages // Handbook of Formal Languages, Vol. 1. - New York: Springer-Verlag New York, 1997. - P. 41–110.

2. Eggan, L.C. Transition graphs and star height of regular events. // Mich. Malh. J. N 10. - 1963. - P. 385–397.

3. Hashiguchi K., Algorithms for determining relative star height and star height // Information and Computation. N 78. - Academic Press, Inc. Duluth, MN, USA, 1988. - P. 124–169..

4. Melnikov, B.F., Vakhitova, A.A. Some more on the finite automata // Korean Journal of Computational and Applied Mathematics. N 5(3). - 1998. - P. 585–595.

5. Stanevichene L.I., Vylitok A.A. An algorithm for transformation of finite automata to regular expressions // Informatica. N 5(3) - 2000. - P. 49–54.

6. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. Том 1. Синтаксический анализ. - М.: Мир, 1978. - 308 с.

7. Евстигнеев В.А. Применение теории графов в программировании. - М.: Наука, 1985. - 352 с.

8. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. - М.: МЦНМО, 2001. - 960 с.

9. Ахо А., Хопкрофт Д.,Ульман, Дж. Построение и анализ вычислительных алгоритмов. - М.:Мир, 1979. - 384 с.

Раздел II

Методы обработки экспериментальных данных

Гришин С.В., Ватолин Д.С., Лукин А.С., Путилин С.Ю., Стрельников К.Н.

Обзор блочных методов оценки движения в цифровых видео сигналах

1. Введение

Цифровое видео сегодня широко распространено, в основном благодаря спутниковому, кабельному и домашнему телевидению, а на территории США аналоговое телевещание планируется прекратить уже в 2009 году. Кроме того, на цифровое видео быстро переходят многие бытовые устройства, такие как видеопроигрыватели (замещение VHS кассет на DVD и MPEG-4 диски), видеокамеры (съемка в цифровые форматы DV и MPEG-2); и даже телевизоры с аналоговой электронно-лучевой трубкой быстро сменяются LCD телевизорами, цифровыми проекционными системами и плазменными панелями. Во всех этих устройствах используются алгоритмы обработки и сжатия видео.

Задача оценки движения является ключевой при работе с цифровым видео. Причиной этому является исключительная важность информации о движении при анализе временной составляющей видеосигнала, которая, в свою очередь, является важнейшей характеристикой видеосигналов и во многом определяет специфику задач цифровой обработки видео. Поясним это на примере задачи сжатия видео.

В первых системах сжатия цифрового видео кадры обрабатывались независимо друг от друга. Каждый кадр кодировался как изображение, а не как часть видео потока. Затем появились алгоритмы, использующие вычитание соседних кадров. В них кодировались не сами кадры, а разница каждого кадра с предыдущим. Этот прием обусловил значительный рост эффективности алгоритмов сжатия благодаря тому факту, что соседние кадры видео потока, как правило, очень похожи, и их разница часто близка к нулю. Исключением из этого правила были случаи наличия движения между

соседними кадрами. Следующим шагом стало появление алгоритмов, использующих компенсацию движения. Компенсацией движения называется преобразование одного из пары кадров, использующее информацию о движении между этими кадрами, и осуществляемое таким образом, чтобы все объекты в кадре имели позиции на момент времени второго кадра пары. Другими словами, компенсация движения делает один из кадров пары максимально похожим на другой, используя информацию о движении между ними. Таким образом, компенсация движения позволяет использовать при сжатии избыточность видео потока во времени даже при наличии движения между кадрами, чего не могли делать алгоритмы сжатия видео предыдущего поколения.

Следует особо выделить две области применения алгоритмов оценки движения (ОД): сжатие и обработка видео. В этих областях к алгоритму ОД предъявляются различные требования. В сжатии видео критическое значение имеет размер информации о движении и скомпенсированной межкадровой разнице (разница между текущим кадром и скомпенсированным). При этом не важно, соответствует ли направление найденных векторов движения реальному движению объектов в видео потоке: главное требование к векторам движения – минимизация скомпенсированной межкадровой разницы. В алгоритмах обработки видео, напротив, объем информации о движении не играет никакой роли, основное значение здесь имеет точность найденной информации о движении, соответствие найденных векторов реальному движению в видео последовательности. Это обусловило появление группы алгоритмов ОД, предназначенных для поиска «истинного» движения (true motion estimation). Специфика указанных требований к алгоритмам ОД очень важна и найдет свое отражение в обзоре алгоритмов, представленной в основной части данной статьи.

В виду особой важности задачи ОД, к настоящему моменту было разработано великое множество соответствующих алгоритмов, которые можно разделить на следующие группы: блочные методы, методы оптического потока, фазовой корреляции, глобальной оценки движения, слежения за особенностями, многокадровой ОД, а также алгоритмы, комбинирующие приемы методов указанных категорий. Наиболее многочисленной из перечисленных является группа блочных методов. Это обусловлено универсальностью, невысокой вычислительной сложностью и сравнительно высокой эффективностью алгоритмов этой категории. Не последнюю роль сыграла также простота их аппаратной реализации. Именно по этим причинам в данной работе представлен обзор блочных алгоритмов ОД.

2. Понятие информации о движении

Цифровое видео представляет собой упорядоченный набор кадров. Именно поэтому применительно к нему часто используют термин видео последовательность. В данной работе также будем придерживаться этой терминологии. Видео последовательность будем обозначать, как $I(t)$, где t – порядковый номер кадра. Каждый кадр – это матрица пикселей, размер этой матрицы обозначим $W \times H$.

Будем использовать следующие обозначения: $p = (x, y)$, $(x, y) \in [0, W-1] \times [0, H-1]$ – вектор координат пикселя в кадре, $I(p,t)$ – яркость пикселя с координатами $p = (x, y)$ в кадре $I(t)$. Вообще говоря, яркость – это только один из цветовых параметров пикселя. Существует множество цветовых моделей (RGB, YUV, Lab и т.д.), в каждой из которых цвет определяется несколькими компонентами. Однако человеческий глаз наиболее чувствителен к яркостной компоненте. По этой причине, а также для простоты изложения, в данной работе будем предполагать наличие только яркостной компоненты (черно-белое видео).

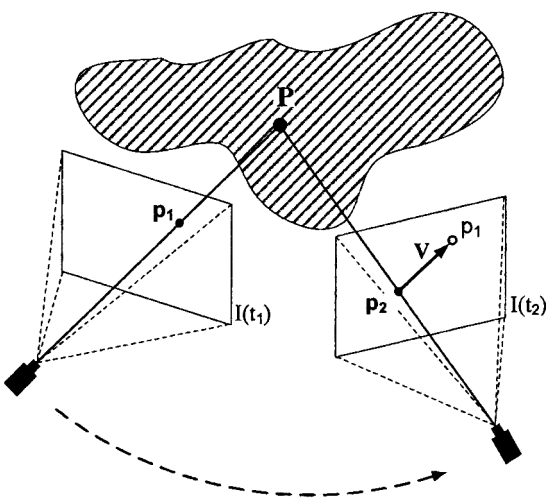


Рис. 1. Схема вычисления вектора движения

Информацией о движении в обработке видео называют двумерный массив *векторов движения*, размер которого равен размеру кадра $W \times H$. При этом под вектором движения в заданной точке

понимается вектор изменения координат этой точки между двумя заданными кадрами. Поясним смысл этого понятия с помощью Рис. 1. Рассмотрим трехмерную сцену без движения и два кадра $I_1 = I(t_1)$, $I_2 = I(t_2)$, полученных при помощи камеры из разных точек. Точке P трехмерной сцены в кадре I_1 соответствует пиксель с координатами p_1, будем его называть образом точки P на кадре I_1. Тогда образом точки P на кадре I_2 будет пиксель с координатами p_2. Сразу заметим, что под образом здесь понимается не само изображение точки P на кадре, а вектор координат ее проекции на матрицу камеры. Это означает, что образ точки P определен даже тогда, когда она ее изображение на кадре отсутствует, например, вследствие наличия какого-либо объекта на переднем плане. Вектор движения V в точке p_2 для пары кадров I_1 и I_2 определяется так:

$$V_{I_1}^{t_2}(p_2) = (u, v) = p_1 - p_2 .$$

В данном примере рассмотрен случай неподвижной сцены и движущейся камеры. В общем случае, изменение координат точек трехмерной сцены на кадрах может быть обусловлено как движением камеры, так и движением объектов сцены.

3. Общая схема алгоритмов блочной оценки движения

Каждый кадр видео последовательности разбивается на множество неперекрывающихся блоков $B_{i,j}$ заданного размера, где i, j – координаты блока. Разбиение производится так, что все блоки покрывают весь кадр, т.е. их суммарная площадь равна площади кадра.

Задача ОД сводится к задаче поиска вектора движения $v_{i,j}$ для каждого блока $B_{i,j}$. При этом вектора $v_{i,j}$ определяются соотношением (1):

$$v_{i,j} = \arg\left(\min_{v_{i,j} \in O}\left(F(t,i,j,v_{i,j})\right)\right), \quad (1)$$

$$O = \left\{(x,y) \mid x \in [-u_{\max}, u_{\max}],\ y \in [-v_{\max}, v_{\max}]\right\}, \quad (2)$$

$$SAD(t,i,j,v) = \sum_{p \in B_{i,j}} \left| I(p,t) - I(p+v,t\text{-}1) \right|, \quad (3)$$

где

O – область поиска векторов движения, $u_{\max}, v_{\max}$ – целые положительные числа;

$F(t,i,j,v_{i,j})$ – функция соответствия блоков, это мера близости блоков текущего и предыдущего кадров. Примером такой функции является SAD (Sum of Absolute Differences), определяемая формулой (3);

Суть работы алгоритмов данной группы заключается в следующем. Для каждого блока текущего кадра производится минимизация функции соответствия блоков по 4-му аргументу, при этом область минимизации может быть любой, единственным ограничением является то, что она должна быть подмножеством области поиска O. В качестве вектора движения для каждого блока выбирается аргумент минимума функции соответствия, вычисленный в этом блоке. Фактически при вычислении функции соответствия производится определение «похожести» двух блоков: блока текущего кадра и блока предыдущего кадра, смещенного на вектор $v_{i,j}$. Таким образом, процесс минимизации функции соответствия является поиском блока предыдущего кадра, наиболее «похожего» на текущий блок.

Важно заметить, что размер области поиска определяет максимальный модуль векторов движений. На практике нередки случаи, когда алгоритм ОД не в состоянии найти верные вектора движения только потому, что амплитуда движения в видео слишком велика.

Для удобства дальнейшего изложения кадр, для блоков которого производится поиск соответствий, будем называть текущим, а кадр, в котором производится поиск – опорным.

4. Базовые методы

В этом разделе будут рассмотрены базовые подходы блочной ОД, такие как полный перебор, шаблонные методы, метод иерархического поиска и методы, использующие вектора-кандидаты. Описание начнем с наиболее простого и очевидного алгоритма полного перебора.

4.1. Алгоритм полного перебора

Поскольку область поиска O конечная, то наиболее очевидным методом минимизации функции соотношения блоков является полный перебор всех значений аргумента $v \in O$.

Данный подход имеет свои достоинства и недостатки. Достоинством данного метода является гарантированное нахождение глобального минимума функции соответствия для каждого блока.

Однако, как было замечено во введении, не во всех приложениях важно найти именно глобальный минимум. В обработке видео критическое значение имеет определение «истинных» векторов движения, независимо от величины соответствующих им значений функции соответствия.

Очевидным недостатком является вычислительная сложность данного метода. Даже в свете высокой мощности современных процессоров, полный перебор может быть неприемлем для обработки в режиме реального времени в случае высокого разрешения видео и большой области поиска.

Логическим продолжением алгоритма полного перебора являются методы шаблонного поиска.

4.2. Методы шаблонного поиска

Данные методы представляют собой дискретные варианты покоординатного спуска. Они основываются на предположении, что функция соответствия достаточно гладкая для каждого блока, без локальных минимумов.

Перед началом описания общей схемы алгоритмов данного класса следует ввести понятие шаблона. Шаблоном является набор координат точек, причем координаты точек отсчитываются от центра шаблона. Таким образом, для произвольно заданной точки по шаблону можно определить набор координат нескольких точек, число этих точек зависит от используемого шаблона.

Поиск вектора в каждом блоке является итеративным процессом. На каждой итерации вычисляется координата центра шаблона, координаты всех точек шаблона, и, затем, значения функции соответствия в каждой из точек шаблона. Центр шаблона на первой итерации называют центром поиска, он обычно равен *(0, 0)*. В качестве центра шаблона для следующей итерации выбирается та точка шаблона, в которой был достигнут минимум функции соответствия. Затем проверяется условия останова поиска, и в зависимости от результата производится переход к следующей итерации или завершение поиска вектора в данном блоке. При этом в качестве результата выбирается вектор, соответствующий точке минимума функции соответствия на шаблоне последней итерации.

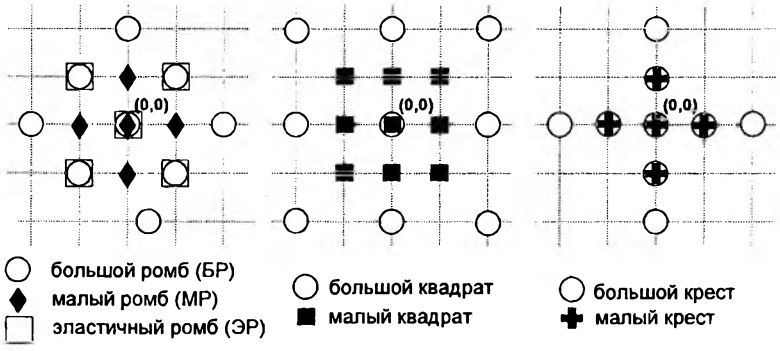


Рис. 2. Примеры шаблонов поиска

Таким образом, шаблонный алгоритм ОД определяется используемым шаблоном. Наиболее часто используемыми на практике шаблонами являются большой и малый ромбы (БР и МР), большой и малый квадраты, а так же большой и малый кресты (см. Рис. 2).

Основным недостатком методов данного класса является их склонность к нахождению локальных минимумов функции соответствия вместо глобальных. В каждом блоке, как правило, выбирается вектор, соответствующий одной из ближайших к центру поиска точек локального минимума функции соответствия, а вовсе не «истинному» движению или ее глобальному минимуму. Однако, у данного класса методов есть существенное достоинство: они значительно сокращают перебор возможных векторов движения, тем самым ускоряя алгоритм.

В современных алгоритмах оценки движения шаблонный поиск, в основном, используется для финального уточнения векторов, полученных на предыдущих шагах. При этом на разных итерациях могут использоваться различные шаблоны. Наиболее интересные примеры алгоритмов данного класса можно найти в статьях [1], [2].

4.3. Иерархический поиск

Идея алгоритмов данной группы заключается в следующем. Перед началом поиска производится вычисление $N-1$ уменьшенных «копий» текущего и опорного кадров, при этом каждая очередная копия в $2n$ (n – натуральное число) раз меньше предыдущей (см. Рис. 3). Пары кадров одинакового размера будем называть уровнями, т.е. на одном уровне опорный и текущий кадры одинакового размера. Тогда все множество пар кадров можно представить N уровнями. Пронумеруем уровни согласно размеру содержащихся в них кадров от меньшего к

большему: 1-й уровень будет содержать кадры минимального размера, *N*-й – кадры исходного размера. Процесс оценки движения состоит из *N* итераций, на каждой из которых обрабатывается пара кадров из уровня с соответствующим номером, т.е. обработка идет от кадров меньшего размера к большему. На каждой итерации производится ОД каким либо из известных методов, например шаблонным поиском. При этом в качестве стартовой точки на каждой итерации выбирается векторное поле, полученное с предыдущей итерации. Другими словами, каждая очередная итерация производит уточнение векторов, вычисленных на предыдущей итерации. При переходе на очередную итерацию размеры области поиска и блоков, для которых оцениваются вектора, обычно увеличиваются в *2n* раз для того, чтобы число блоков в кадре на каждой итерации не менялось.

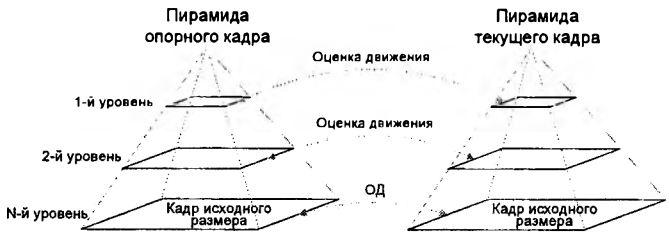


Рис. 3. Схема иерархических уровней

Достоинством алгоритмов данной группы является перебор сокращенного числа векторов, т.е. повышенная вычислительная эффективность. Однако это достоинство нивелируется в случае использования постоянного размера блоков и областей поиска при переходе с одного уровня на другой. Тем не менее, есть аргумент в пользу того, чтобы фиксировать размер блоков и областей для всех итераций. При фиксированном размере блоков повышается устойчивость векторов в гладких областях, поскольку вероятность попадания контрастных деталей возрастает вместе с ростом площади блока. Это позволяет частично решить одну из основных проблем блочных методов ОД – проблему поиска векторов в гладких областях. Для блоков из таких областей функция соответствия принимает очень близкие (или даже равные, в случае абсолютно гладких областей) значения для различных векторов движения, т.е. эффективность использования ее значений в качестве основного критерия выбора вектора сводится к минимуму. Повысить вероятность успешного нахождения вектора движения позволяет использование блоков большего размера, но это эффективно лишь в случаях, когда размер блока больше размера гладких областей. Возвращаясь к иерархическому

подходу, можно сказать, что выполнение ОД при фиксированном размере блока на уровнях с уменьшенными кадрами позволяет повысить вероятность попадания в блок контрастных деталей (и тем самым увеличить вероятность успешного нахождения вектора) по сравнению со случаем изменения размера блока при переходе на следующий уровень.

Постоянный размер области поиска позволяет увеличить максимальную амплитуду векторов движения по сравнению со случаем, когда размер области поиска уменьшается вдвое при переходе на очередной уровень.

Дополнительным преимуществам данного метода является устойчивость к шуму, поскольку во время уменьшения изображений, как правило, удаляются высокочастотные шумы. Однако вместе с высокочастотными шумами могут пропасть и мелкие детали, что приведет к неправильному определению движения в детализированных областях.

Методы данного класса находят широкое применение во многих задачах обработки видео [3], [4]. В работе 3 описывается оригинальный способ уменьшения вычислительной сложности иерархического поиска. Авторы предложили две идеи: улучшенный критерий определения неверно найденных векторов и метод выбора начального уровня иерархии. Уменьшение сложности определения «плохих» векторов достигается за счет использования функции соответствия блоков с меньшей вычислительной сложностью по сравнению с SAD. Начальный уровень иерархии выбирается на основе предположения о близости значений функции соответствия для векторов из небольшой окрестности. Авторы предлагают выбирать начальный уровень иерархии на основе анализа данных, полученных при вычислении значений функций соответствия для векторов из небольшой окрестности.

4.4. Методы, использующие векторы-кандидаты

Для большинства видео последовательностей справедливо утверждение, что вектора движения соседних блоков очень похожи, так как эти блоки зачастую принадлежат одному движущемуся объекту. Это утверждение привело к появлению целого класса методов ОД, использующих векторы-кандидаты.

Основная идея алгоритмов этой группы очень проста. Перед вычислением информации о движении для текущего блока формируется набор, состоящий из уже вычисленных векторов движения соседних блоков. При этом соседние блоки могут выбираться как в пространственной области, так и во временной. Сформированный набор

векторов называется набором кандидатов. В качестве вектора движения в каждом блоке выбирается лучший вектор из набора кандидатов. В качестве критерия поиска обычно используется функция соответствия. Наиболее яркими представителями алгоритмов данной группы являются 3DRS [5] и E3DRS (Enhanced 3DRS, [6]).

Метод 3DRS (3D recursive search) формирует набор векторов-кандидатов из найденных векторов движения со смещениями (–1, –1) и (1, –1) в текущем кадре и (–2, 2), (2, 2) в предыдущем кадре. К первым двум векторам-кандидатам прибавляется равномерно распределенный случайный вектор с амплитудой до ±3 пикселей. После этого из полученных кандидатов выбирается вектор с наименьшей SAD. Использование векторов-кандидатов, взятых с различных направлений, позволяет методу 3DRS достаточно быстро сходиться к реальному направлению движения вблизи границ объектов, по сравнению с более простыми методами.

В методе E3DRS используется похожий набор векторов-кандидатов, однако здесь имеется стадия дополнительного поиска вектора с наилучшей SAD по шаблону «малый квадрат» (см. Рис. 2) вокруг выбранного вектора-кандидата. Это обеспечивает лучшие величины SAD, чем у метода 3DRS.

Методы, использующие векторы-кандидаты, часто имеют низкую вычислительную сложность, но при этом обеспечивают гладкость векторного поля, что делает их пригодными для использования в аппаратуре реального времени.

5. Комбинированные методы

В большинстве современных блочных алгоритмов нахождения движения используются различные комбинации базовых подходов, описанных выше.

Наиболее популярной комбинацией является совместное использование подхода, использующего векторы-кандидаты, и шаблонного поиска. Идея методов данной группы состоит в уточнении лучшего вектора набора с помощью шаблонного поиска. Благодаря простоте и вычислительной эффективности алгоритмы данной группы достаточно часто становятся предметом интереса исследователей ([7]-[9]). В качестве примера рассмотрим наиболее популярный из современных представителей данной группы алгоритм FAME (Fast Adaptive Motion Estimation), описанный в статье [9].

Этот алгоритм использует приемы для быстрого определения неподвижных блоков, повышения устойчивости поиска векторов в гладких областях, раннего останова поиска для экономии вычислительных ресурсов, а также повышения эффективности

использования наборов кандидатов и шаблонов. В рамках данной работы наибольший интерес представляют последняя пара приемов. Рассмотрим их более подробно.

Повышение эффективности использования наборов кандидатов достигается за счет добавления в набор дополнительных векторов. Стандартный набор включает в себя вектора из 3 блоков, находящихся слева, сверху и справа сверху относительно текущего блока. Помимо них в набор добавляются еще два вектора. Первый вычисляется как среднее значение векторов стандартного набора. Второй называется инерционным кандидатом и равен вектору того блока предыдущего кадра, проекция которого на текущий кадр имеет наибольшее пересечение с текущим блоком (см. Рис. 4). При этом проецирование осуществляется вдоль вектора предыдущего кадра. Стоит заметить, что имеется в виду именно предыдущий кадр, а не опорный, т.е. инерционный вектор выбирается из векторного поля, вычисленного алгоритмом ОД для предыдущего кадра. Фактически, этот прием использует предположение о равномерном движении объектов, а сам инерционный кандидат является продолжением траектории движения блока предыдущего кадра. Инерционный кандидат также используется в алгоритмах, описанных в статьях [9], [11], [12]. Наличие указанных двух дополнительных кандидатов позволяет повысить точность поиска векторов.

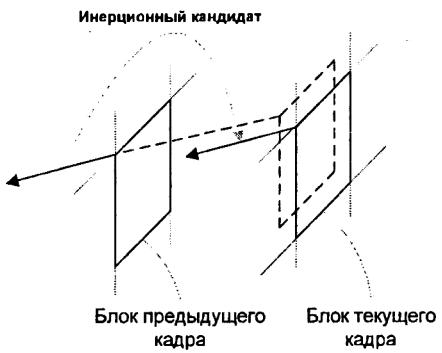


Рис. 4. Схема вычисления инерционного кандидата

Новизна в использовании шаблонных методов заключается в следующем. Алгоритм использует шаблоны ББ, МБ и шаблон «эластичный ромб» (ЭР, см. Рис. 2). Использование этих шаблонов при поиске вектора в каждом блоке зависит от нескольких характеристик локальной окрестности текущего блока. В частности, используется величина, характеризующая гладкость векторного поля, и ошибки

компенсации соседних блоков. В зависимости от условий, справедливых для этих величин, производится выбор одной из 3 стратегий поиска, при этом в рамках каждой стратегии могут быть использованы не все 3 шаблона. Полное описание стратегий поиска можно найти в работе [9].

6. Заключение

Алгоритмы каждой из рассмотренных групп имеют свои достоинства и недостатки. Ни один из базовых методов, использованных в чистом виде, не дает приемлемых результатов на практике. Для построения эффективного метода оценки движения необходимо комбинировать приемы, использованные в алгоритмах различных типов, так чтобы недостатки одного метода компенсировались достоинствами другого. Примером такого метода является алгоритм FAME [9], сочетающий в себе использование векторов кандидатов, методов шаблонного поиска, условий раннего останова поиска, способов быстрого определения неподвижных блоков и др.

Блочные алгоритмы оценки движения являются весьма выгодным компромиссом по соотношению вычислительная сложность/точность найденных векторов. Комбинирование приемов из алгоритмов различных категорий в рамках класса блочных методов позволяет построить универсальные алгоритмы ОД, обладающие заданными свойствами и легко реализуемые аппаратно.

7. Литература

1. S. Zhu and K. K. Ma, "A new diamond search algorithm for fast blockmatching motion estimation," IEEE Trans. Image Processing, vol. 9, pp. 287–290, Feb. 2000.

2. Xuan Jing and Lap-Pui Chau, "An Efficient Three-Step Search Algorithm for Block Motion Estimation", IEEE Transactions on Multimedia, Vol. 6, N. 3, June 2004.

3. Lopes F., Ghanbari M., "Hierarchical motion estimation with spatial transforms", in proceedings of IEEE International Conference on Image Processing, Vol. 2, pp. 558-561, Vancouver, BC, Canada, 2000.

4. Tae Gyoung Ahn, Yong Ho Moon, Jae Ho Kim, "Fast Full-Search Motion Estimation Based on Multilevel Successive Elimination Algorithm", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 14, N. 11, November 2004.

5. G. de Haan, P.W.A.C Biezen, H. Huijgen, and O.A. Ojo, "True Motion Estimation with 3-D Recursive Search Block-Matching", IEEE

Transactions on Circuits and Systems for Video Technology, Vol. 3, October 1993, pp. 368-388.

6. S. Olivieri, L. Albani, and G. de Haan, "A low-complexity motion estimation algorithm for H.263 video coding", Proc. Philips Conf. on Digital Signal Processing, Nov. 1999, paper 17.3, Veldhoven (NL).

7. P. I. Hosur and K. K. Ma, "Motion vector field adaptive fast motion estimation", presented at the Second Int. Conf. Inf., Commun., Signal Process., Singapore, Dec. 1999.

8. M. Tourapis, O. C. Au, and M. L. Liou, "Predictive Motion Vector Field Adaptive Search Technique (PMVFAST)," presented in ISO/IEC JTC1/SC29/WG11 MPEG2000, Noordwijkerhout, NL, March 2000.

9. Ishfaq Ahmad, Weiguo Zheng, Jiancong Luo, Ming Liou, "A Fast Adaptive Motion Estimation Algorithm", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 16, N. 3, March 2006.

10. Shih-Yu Huang, Chuan-Yu Cho, and Jia-Shung Wang, "Adaptive Fast Block-Matching Algorithm by Switching Search Patterns for Sequences With Wide-Range Motion Content", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 15, N. 11, November 2005.

11. Yongfang Liang, Ishfaq Ahmad, Jiancong Luo, Yu Sun, "On Using Hierarchical Motion History for Motion Estimation in H.264/AVC", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 15, N. 12, December 2005.

12. С. Путилин, "Быстрый алгоритм нахождения движения в видеопоследовательностях", Труды конференции Graphicon-2006, с. 407-410, Новосибирск, Академгородок, Россия, Июль 2006.

Куликов Д. Л.

Временной метод маскирования искажений в видео на основе обработки оптического потока[1]

1. Введение

Вычислительные мощности современных компьютеров постоянно растут, в связи с этим появляются новые стандарты кодирования видео, все сильнее использующие внутренние зависимости в видеоматериале и, как следствие, уменьшающие размер закодированного видео. При этом возникает следующая проблема при ошибках передачи закодированного сигнала – возрастает количество искажений, вызванных ошибками, так как растёт количество информации, передаваемой в каждом бите закодированного видео. Поэтому наравне с исследованиями в области новых методов помехоустойчивого кодирования активно развиваются методы борьбы с искажениями. Поскольку в общем случае восстановить потерянную в результате ошибок передачи кодированного сигнала информацию не возможно, исследователями ставится задача маскирования искажений, возникающих в видео.

2. Предложенный метод

Существуют две различные задачи маскирования искажений в видео: при работе в режиме реального времени в процессе декодирования; и при постобработке видеоматериала, когда скорость работы становится не столь критичной, при этом возрастают требования к визуальному качеству обработки. Предложенный метод относится к классу методов постобработки, при этом, если провести его оптимизацию, или использовать большие вычислительные мощности, возможен переход в класс методов работы в режиме реального времени.

Основываясь на том факте, что большинство современных видео кодеков во время процесса декодирования обнаруживают синтаксически и семантически неверные конструкции, выполняя процесс локализации артефактов, вызванных этими ошибками, можно предположить, что на выходе этого процесса имеется маска с обнаруженными артефактами. Учитывая также последовательность процесса декодирования, входными данными для алгоритма маскирования могут быть искаженный кадр, маска артефактов и в

[1] Работа выполнена при поддержке гранта РФФИ № 07-01-00759-а

общем случае предыдущие декодированные кадры. Для упрощения системы в качестве предыдущих кадров используется только один кадр.

На рисунке 1 показана схема предложенного метода. Искаженный кадр, предыдущий кадр и маска ошибок подаются на вход алгоритму построения оптического потока, описанному в разделе 2.1. Данный алгоритм находит вектор движения для каждого пикселя из окрестности неизвестной области. Для построения оптического потока может использоваться любой алгоритм, обладающий приемлемой точностью, в том числе и алгоритмы, основанные на первоначальном приближении векторов для каждого пикселя при помощи вектора для блока пикселей, который доступен декодеру. В предложенном алгоритме в основе используется метод Лукаса-Канаде. Далее полученное поле векторов фильтруется с целью отсечения ошибочно найденных векторов. Данный блок описан в разделе 2.2. После фильтрации происходит поиск неоднородностей (разрывов) векторного поля, описанный в разделе 2.3. После фильтрации осуществляется интерполяция разрывов на внешней границы искаженной области внутрь области, после чего производится реконструкция векторов движения в соответствии с найденными и интерполированными неоднородностями. Данный блок описан в разделе 2.4. После реконструкции векторов движения для каждого пикселя искаженной области происходит компенсация движения – то есть каждый пиксель текущего кадра внутри области заменяется на пиксель опорного кадра в соответствии с вектором движения.

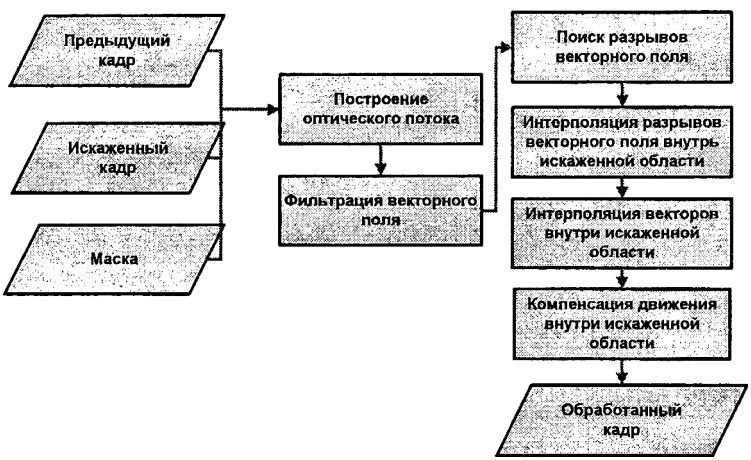


Рисунок 1: Блок-схема предложенного алгоритма

2.1. Построение оптического потока

Задача построения оптического потока состоит в нахождении вектора движения (то есть, фактически прототипа данного пикселя в опорном кадре) для каждого пикселя в кадре либо области кадра. Для решения этой задачи в данном алгоритме построения оптического потока за основу взят метод Лукаса-Канаде [1]. Это двухкадровый дифференциальный метод, в основе которого лежит предположение о неизменности или слабом изменении интенсивности или цвета пикселя между двумя кадрами. То есть выполняется условие:

$$I(x,y,t) = I(x + \delta x, y + \delta y, t + \delta t), \quad (1)$$

где $I(x,y,t)$- яркость пикселя с координатами (x,y) в момент времени t. Далее правая часть уравнения (1) может быть представлена при помощи ряда Тейлора

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t + \ldots \quad (2)$$

Откуда следует, что

$$\frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t = 0 \quad (3)$$

$$\frac{\partial I}{\partial x}\frac{\delta x}{\delta t} + \frac{\partial I}{\partial y}\frac{\delta y}{\delta t} + \frac{\partial I}{\partial t}\frac{\delta t}{\delta t} = 0 \quad (4)$$

Что приводит к следующему уравнению

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t}\delta t = 0, \text{ где} \quad (5)$$

V_x, V_y – x,y –компоненты скорости оптического потока $I(x,y,t)$, а $\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}$ и $\frac{\partial I}{\partial t}$ производные изображения в точке (x,y,t) в соответствующих направлениях. Уравнение можно переписать в виде

$$\nabla I \cdot \vec{V} = -I_t \quad (6)$$

Для решения этого уравнения с двумя неизвестными, необходимо вести дополнительные ограничения на оптический поток. Таким ограничением может являться следующее – поток (V_x, V_y) можно считать постоянным в окне mxm пикселей, где $m>1$, нумеруя пиксели $1\ldots n$, $n=m^2$ получается следующая система уравнений

$$\begin{aligned} &I_{x1}V_x + I_{y1}V_y = -I_{t1} \\ &\ldots \\ &I_{xn}V_x + I_{yn}V_y = -I_{tn} \end{aligned} \quad (7)$$

Получается переопределенная система

$$\begin{bmatrix} I_{x1} & I_{y1} \\ \ldots & \\ I_{xn} & I_{yn} \end{bmatrix} \begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} -I_{t1} \\ \ldots \\ -I_{tn} \end{bmatrix} \quad (8)$$

$$A\vec{v} = -b \tag{9}$$

Для решения используется метод наименьших квадратов:

$$A^T A\vec{v} = A^T(-b) \tag{10}$$

$$\vec{v} = (A^T A)^{-1} A^T(-b)$$

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum I_{xi}^2 \sum I_{xi} I_{yi} \\ \sum I_{xi} I_{yi} \sum I_{yi}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum I_{xi} I_u \\ -\sum I_y I_u \end{bmatrix} \tag{11}$$

Для ускорения работы метода вычисления производных проводится для разных разрешений – сначала для грубого приближения, затем производятся уточнения. На выходе алгоритма получается оптический поток. Для поставленной задачи необходимо вычислить оптический поток только вблизи границы неизвестной области.

2.2. Фильтрация векторного поля

Поскольку в результате использования метода Лукаса-Канаде для построения оптического потока могут возникнуть ошибочно найденные вектора – что особенно характерно для гладких областей изображения, где отсутствуют контрастные переходы и изменения, актуальной является задача фильтрации полученного векторного поля с целью отсечения ошибочно найденных векторов и одиночных векторов выбросов. Для фильтрации используется функция доверия вектору, которая строится на основе вычисления ошибки компенсации для данного вектора, то есть того, насколько точно он представляет соответствующий пиксель, и на основе однородности поля векторов.

Сначала вычисляется ошибка компенсации ξ_i для каждого вектора – для ее вычисления используется шаблон поиска с четырьмя пространственными соседями для пикселя $\theta(x_i, y_i)$

$$\xi_i = \sum_{(x_j, y_j) \in \theta(x_i, y_i)} \left| I(x_j, y_j, t) - I(x_j + mv_i^x, y_j + mv_i^y, t-1) \right| \tag{12}$$

После вычисления ошибки компенсации вычисляется оценка однородности поля в данном пикселе – для этого вычисленное поле разбивается на кластеры однородности, то есть на пространственно связанные области, в которых вектора принадлежат одному объекту. Далее вычисляется следующая оценка доверия вектору – для кластеров размера 1, то есть одиночных векторов, не вошедших ни в один из кластеров, по формуле:

$$\sigma_i = \sum_{j \in \theta(i), |C_j| > 1} \sqrt{(mv_i^x - mv_j^x)^2 + (mv_i^y - mv_j^y)^2} \Big/ \sum_{j \in \theta(i), |C_j| > 1} 1 \quad , |C_i| = 1 \tag{13}$$

То есть в оценке однородности участвуют только соседние вектора, принадлежащие не единичным кластерам. Для кластеризации используется модификация накапливающего иерархического алгоритма кластеризации [2], где на каждом шаге алгоритма объединяются кластеры, являющиеся пространственными соседями, то есть обладающими минимальным расстоянием между собой, как в алгоритме [3], и это расстояние не превышает порогового значения.

$$\rho(C_1, C_2) = \frac{1}{|C_1| + |C_2|} \sum_{mv_i \in C_1} \sum_{mv_j \in C_2} \sqrt{(mv_i^x - mv_j^x)^2 + (mv_i^y - mv_j^y)^2} \qquad (14)$$

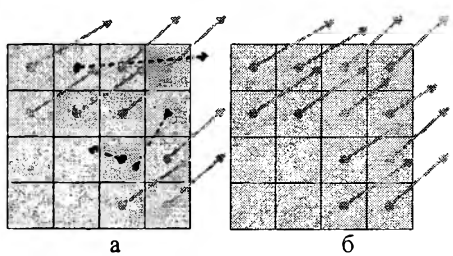
После этого решается вопрос о принадлежности вектора множеству ошибочно найденных векторов на основе сравнения средневзвешенной оценки доверия со средней оценкой для соседних векторов:

$$mv_i \in \Omega : \alpha\xi_i + \beta\sigma_i \le \omega(\alpha\bar{\xi} + \beta\bar{\sigma}) \qquad (15)$$

После нахождения ошибочных векторов, каждый такой вектор заменяется на средневзвешенную сумму соседних правильно найденных векторов, где весовые коэффициенты обратно пропорциональны ошибки компенсации векторов, вычисленной по формуле (12):

$$mv_i^{new} = \frac{\sum_{j \in \theta(i), \xi_j \le \bar{\xi}} mv_j / \xi_j}{\sum_{j \in \theta(i), \xi_j \le \bar{\xi}} 1/\xi_j} \qquad (16)$$

Пример работы такого алгоритма показан на Рисунке 2.



а б

Рисунок 2: Пример работы алгоритма фильтрации векторов а – до фильтрации, б – после фильтрации

2.3. Поиск неоднородностей векторного поля

После фильтрации происходит поиск неоднородностей (разрывов) векторного поля. Для этого используются как оценка расстояния между векторами, оценка расстояния в цветовом пространстве, так и оценка доверия векторам. На начальном этапе работы алгоритма выбираются вектора, чья оценка доверия выше порового значения.

$$\omega(mv_i) = \frac{1}{\sum_{(x_j,y_j)\in\theta(x_i,y_i)} \left| I(x_j,y_j,t) - I(x_j + mv_i^x, y_j + mv_i^y, t-1) \right|} \geq \overline{\omega} \quad (17)$$

где порог $\overline{\omega}$ вычисляется адаптивно и зависит от дисперсии яркости в кадре – таким образом, для зашумленного или текстурного видео данный порог доверия будет выше.

Далее для векторов с оценкой доверия выше порового значения проводится оценка расстояния между векторами и происходит сравнение со средним значением среднеквадратичного отклонения, вычисленным для всех найденных на предыдущем шаге кластеров векторов:

$$\rho(mv_i, mv_j) = \sqrt{(mv_i^x - mv_j^x)^2 + (mv_i^y - mv_j^y)^2} \geq \overline{\rho} \quad (18)$$

$$\overline{\rho} = \frac{1}{n}\sum_{i=1}^{n} \sigma(C_j)$$

После этого аналогичным образом происходит оценка расстояния в цветом пространстве. Если вектор mv_i имеет начало в точке (x,y), а векторов mv_j – в точке $(x+1,y)$, а $I(x,y)$ – это яркость (цветность) в точке с координатами (x,y), то данная оценка вычисляется по формуле:

$$\rho_{color}(mv_i, mv_j) = \sqrt{(I(x,y) - I(x+1,y))^2} \geq \overline{\rho}_{colur} \quad (19)$$

Вектора, превышающие данную оценку, считаются находящимися на границах разрыва векторного поля. На Рисунке 3 показан пример такого разрыва.

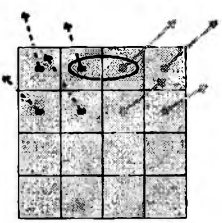


Рисунок 3: Пример неоднородности векторного поля

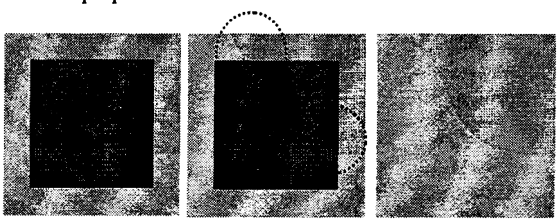
2.4. Интерполяция неоднородностей векторного поля и реконструкция векторов

После нахождения разрывов векторного поля вокруг неизвестной области происходит поиск пар разрывов, обладающих следующими свойствами:

- разрывы находятся на разных сторонах области, то есть неоднородность проходит через область;

- данные разрывы заметны, то есть оценка расстояния между векторами превышают заданный порог, и оценка расстояния в цветовом пространстве также превышает порог;
- эти разрывы сопоставимы, то есть являются парными – у них совпадают расстояния между векторами и яркости (цветности) границ.

Далее происходит интерполяция разрывов внутрь неизвестной области и классификация подобластей к одному из объектов, соответствующих разным частям неоднородности. Для интерполяции возможно использование, как линейной интерполяции, так и более высоких порядков (например, кубической). На Рисунке 4 показаны этапы работы алгоритма интерполяции разрывов, разными цветами обозначены вектора разных объектов.



а б в

Рисунок 4: Этапы работы алгоритма интерполяции разрывов а – нахождение пары разрывов, б – интерполяция внутрь области, в – классификация подобластей

После классификации точек неизвестной области (отнесения их к одному из двух объектов, участвующих в разрыве) происходит интерполяция векторов внутрь области. Для этого используется взвешенная сумма известных соседних векторов. При этом учитывается как близость начала известного вектора неизвестной точке области, так и совпадение или разница классов принадлежности данных точек. Для интерполяции область разбивается на четыре подобласти и в каждой из них восстановление векторов происходит независимо, это сделано для того, чтобы исключить влияние векторов, находящихся на большом расстоянии от точки области:

$$mv_i^1 = \frac{\sum_{j \in \theta(i)} mv_j^1 / \rho(mv_i^1, mv_j^1)\omega(class_i, class_j)}{\sum_{j \in \theta(i)} 1/\rho(mv_i^1, mv_j^1)\omega(class_i, class_j)} \qquad (20)$$

Индекс 1 для mv_i^1 и mv_j^1 означает, что вектора принадлежат одной подобласти 1 (см. Рисунок 5), $\omega(class_i, class_j)$ определяет величину штрафа, когда вектора принадлежат различным классам (разным объектам), например следующим образом:

$$\omega(class_i, class_j) = \begin{cases} 1, i = j \\ 10^4, i \neq j \end{cases} \quad (21)$$

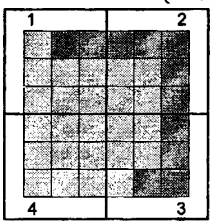


Рисунок 5: Пример разбиения неизвестной области на четыре подобласти

После того, как для каждой пикселя неизвестной области построен вектора движения, происходит реконструкция данного пикселя – для этого происходит замена на пиксель предыдущего кадра в соответствии с вектором, в случае дробных значений вектора движения замена происходит с дополнительной интерполяцией.

3. Результаты экспериментов

Для эксперимента было выбрано 6 видео последовательностей разрешения 352x288 с различными типами движения. В каждую из видео последовательностей были внесены искажения, и точная маска искажений подавалась на вход алгоритмам. Предложенный метод был сравнен с известными алгоритмами временного восстановления такими как:

- BMA[5] и OBMA[6], которые используют геометрическую похожесть внешних границ искаженной области;
- алгоритм сопоставления структур SAA[7];
- MDBM[8], выселяющий похожесть вдоль основных направлений;
- STBMA[9] использующим пространственную и временную похожесть внешних областей
- методом интерполяции векторов в блоках ME [9].

Для объективного сравнения была выбрана метрика SSIM [4] (индекс структурной похожести). Эта метрика имеет более высокую корреляцию с субъективными метриками качества видео, чем широко используемая метрика среднеквадратичного отклонения PSNR, и дает представление о качестве метода в сравнении с другими методами. В качестве результата рассматривались значения метрики для маскированного видео и исходного неискаженного видео. Чем выше значение метрики, тем более похоже маскированное видео на исходное.

На Рисунке 6 показано сравнение методов маскирования, в качестве метрики выбрана delta-SSIM, то есть разница значения метрики и минимального значения метрики для всех методов.

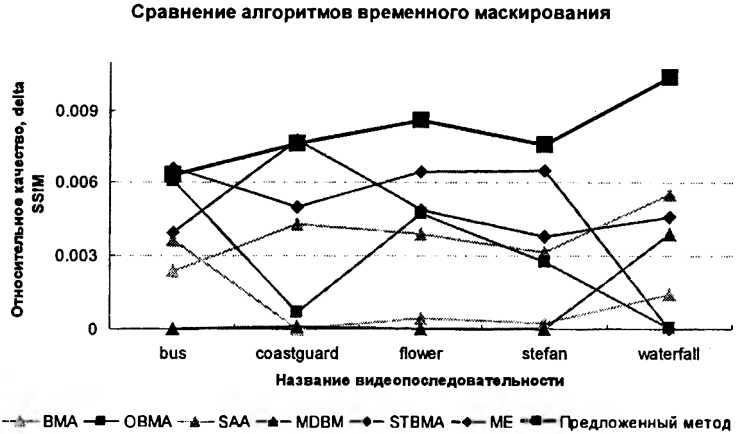
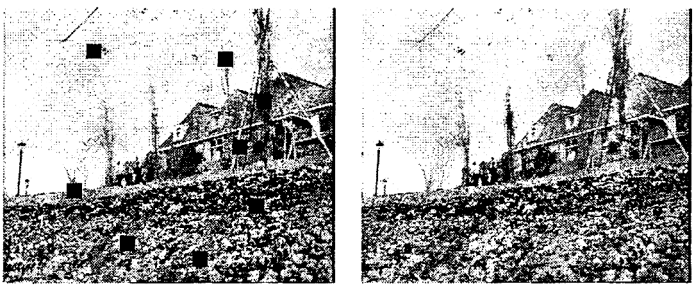


Рисунок 6: Сравнение методов временного маскирования

На Рисунке 7 показан результат работы предложенного метода.



Искаженный кадр Обработанный кадр

Рисунок 7: Результат работы предложенного метода

Также для оценки качества работы предложенного метода было проведено субъективное сравнение по методике SAMVIQ от EBU с участием 10 экспертов, которое показало преимущество предложенного метода над остальными. Результаты можно увидеть на Рисунке 8.

Рисунок 8: Субъективное сравнение временных методов маскирования

4. Заключение

В данной статье предложен новый временной метод маскирования ошибок. Он основывается на подходе с использованием оптического потока и поиска неоднородностей, что позволяет более точно проводить реконструкцию векторов движения в неизвестной области, что в результате приводит к более высокому визуальному качеству маскирования. Предложенный метод может быть использован после или во время этапа декодирования после потерь пакетов во время передачи. При использовании данного метода на этапе декодирования вычислительная сложность может быть дополнительно снижена за счет использования информации, получаемой декодером, такой как, например вектора движения для соседних к искаженному макроблоков для начального этапа построения оптического потока.

Предложенный метод показывает лучшее качество по сравнению с известными методами временного маскирования и он также может использоваться для восстановления видео для удаления царапин, пятен или нежелательных объектов из видео при обработке.

5. Литература

1. B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in IJCAI81, 1981, pp. 674-679.

2. R. Duda, P.E. Hart, D.G. Stork. Pattern Classification. 2nd ed., Wiley-Interscience, 2001.

3. K. Simonyan, S. Grishin, and D. Vatolin, "Confidence Measure for Block-Based Motion Vector Field", in Proc. Graphicon, pp. 110-113, Moscow, June 2008

4. Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, Apr. 2004.

5. J. Zhang, J. F. Arnold, and M. R. Frater, "A cell-loss concealment technique for mpeg-2 coded video," Circuits and Systems for Video Technology, IEEE Transactions on, vol. 10, no. 4, pp. 659-665, 2000.

6. T. Thaipanich, Wu, and J. C. C. Kuo, "Low-complexity mobile video error concealment using obma," December 2007.

7. A. S. Bopardikar, O. I. Hillestad, and A. Perkis, "Temporal concealment of packet-loss related distortions in video based on structural alignment," in In Proceedings of the Eurescom summit, Heidelberg, Germany, April 2005.

8. T.-Y. Kuo and S.-H. Li, "Hybrid temporal-spatial error concealment technique for video communications," in Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on, vol. 3, 2004, pp. 1743-1746 Vol.3.

9. Y. Chen, Y. Hu, O. C. Au, H. Li, and C. W. Chen, "Video error concealment using spatio-temporal boundary matching and partial differential equation," Multimedia, IEEE Transactions on, vol. 10, no. 1, pp. 2-15, 2008.

10. Д. Куликов, Д. Ватолин, "Обнаружение и заполнение статических инородных областей в видео на примере удаления логотипов и сбоев при ошибках передачи", материалы девятого научно-практического семинара "Новые информационные технологии в автоматизированных системах", стр. 13-23, Москва, март 2006.

Попова Е.А.
Московский Государственный Университет

Алгоритм построения карт электрической активности мозга на основе обработке сигнала ЭЭГ и его применение для исследования нейрофизиологических проблем восприятия

1. Введение

Статья посвящена описанию алгоритмов построения пространственно-усредненных карт активности нейронных дипольных источников, и анализу результатов применения данных алгоритмов для обработки экспериментальных данных при исследовании нейрофизиологических проблем визуального восприятия.

В последнее время происходит бурное внедрение методов автоматизированной обработки сигналов и программных технологий для решения актуальных задач биомедицины. Одно из важных направлений биомедицины связано с обработкой сигналов, измеряемых при изучении человеческого мозга. Особую роль в изучении человеческого мозга играют исследования электрической активности методом анализа электроэнцефалограммы (ЭЭГ). Ключевое место в этой проблеме занимает задача локализации источников. Определение зон активности открывает новые возможности для проведения диагностирования и лечения заболеваний, исследования реакций мозга не внешние воздействия, исследование проблем восприятия человеком внешнего мира.

Изучением понимания принципов обработки мозгом чувствительной (сенсорной) информации, последующим ее отражением в память, и сопоставление этой информации с предъявляемыми вновь образами, занимается нейрофизиология. В нейрофизиологических исследованиях на основе ЭЭГ проводятся важнейшие эксперименты по выявлению реакции мозга на поступающие образы. Получая оценку функционального состояния работы мозга, нейрофизиологи выявляют принципы и механизмы внутреннего взаимодействия компонент мозга. Практическое внедрение новых программных систем, на основе новых информационных технологий позволяет дать дополнительные новые инструменты для понимания нейрофизиологами проблем восприятия человеком внешнего мира.

Целью проведения экспериментов регистрации сигнала ЭЭГ как правило является выявление электрических характеристик мозга, связанных с реакцией мозга на определенную группы внешних по

отношению к испытуемому событий, или с внутренним состоянием испытуемого, соответсвующим его *нормальной* активности. Например, внешняя зрительная стимуляция широко применима при исследовании общих характеристик функциональных состояний мозговой деятельности, связанных с распознаванием образов, общением человека, реакциями мозга на неспецифические внешние стимулы (вспышка света, цветовые гаммы). Правильная постановка эксперимента играет очень важную роль в получении достоверных данных о реакциях мозга. Человеческий мозг представляет собой сложную структуру, состоящую из 10^{14} нейронов со множественными связями, судить о которых можно только на уровне регистрируемых следсвтий мозговой активности. Получаемые данные в виде сигнала ЭЭГ содержат в себе множество артефактов, связанных с мышечной активностью, и отражают любые смены внутреннего состояния человека. Следствием использования неинвазивного метода получения данных, являются многократные повторения одних и тех же стадий эксперимента.

Методы математической статистики широко применяются для нахождение общих закономерностей электрических характеристик обработки данных. Среди существующих программных систем локализации активных нейронных источников распространены методы дисперсионного и корелляционного анализа временных последовательностей потенциала при первичной обработки данных, и их последующая связь с результатами локализации. Следует отметить, что статистический анализ полученных зон активности также имеет огромное значение. Соотношение сигнал/шум в исходных данных значительно влияет на появление в рузельтате ошибочных локализаций, связанных с артефактами. В данный работе предлагается метод построения усредненных взвешенных карт активности нейронных источников, основанный на эвристике задачи локализации. Использование программной компоненты, основанной на методе картирования по входной выборке сигнала, дает возможность исследователю (пользователю программной системы) отслеживать характер активности на разных временных масштабах для определенной группы испытуемых (стимулов).

2. Алгоритм построения пространственно-усредненных карт активности нейронных дипольных источников

Пусть входными данными для обработки является набор сигналов ЭЭГ однинаковой размерности. Для каждого временного момента по измеренноым значениям потенциала получены координтанты положения

и силы дипольных источников. В работе [1] описан алгоритм локализации дипольных источников для одного временного момента. Найденные дипольные источники образуют временную последовательность параметров активности соответсвенно временной размерности входного сигнала. Для набора сигналов ЭЭГ входными данными для алгоритма построения усредненных карт будет множество последовательностей временных последовательностей найденных параметров дипольных исчтоников. Так как локализация источников в каждый момент времени проводится независимо, то временной процесс изменения положения диполя можно представить как случайный процесс. Нахождение усредненной области положения диполя-источника для некоторого временного промежутка (временного окна) позволяет определить область стационарности процесса и отследить изменения положения, вызванные некоторым событием.

Сформулируем общий алгоритм формирования пространственных карт активности для группы сигналов ЭЭГ в фиксированном временном окне τ_0. Пусть дана группа N файлов с записью ЭЭГ сигналов одинаковой длины , и T - число измерений потенциала на поверхности головы, сделанное за время τ_0. Предположим, что в каждый временной отсчет были найдены параметры активного источника-диполя. Обозначим через $I = \{I_1, I_2, ..., I_N\}$ множество последовательностей вида

$$I_i = (\{(r_t, \vartheta_t, \varphi_t, v_{r_t}, v_{\vartheta_t}, v_{\varphi_t})^t\}), (1)$$

где i - номер файла ЭЭГ в рассматриваемой группе, $t = 1..T$, T -- число анализируемых временных отсчетов, $(r_t, \vartheta_t, \varphi_t, v_{r_t}, v_{\vartheta_t}, v_{\varphi_t})^t$ - параметры найденного диполя для временного отсчета t. Тогда множеству I поставим в соотвествие новое множество

$$I^K = \{I_1^{K_1}, I_2^{K_2}, ..., I_N^{K_N}\}, (2)$$

где $K_1 ... K_N$ - некоторое числа, которые могут принимать целые значения в интервале $[1..T]$, а

$$I_i^{K_i} = \{(r_j, \vartheta_j, \varphi_j, v_{r_j}, v_{\vartheta_j}, v_{\varphi_j})^j, C_j, P_j\}^K \qquad (1)$$

- упорядоченное по всем C_j множество, то есть $C_j > C_{j+1}$, $j = 1...U_i^{K_i}$, где $U_i^{K_i}$ - количество всех неповторяющихся возможных

положений $(r_i, \vartheta_i, \varphi_i)$ диполя для всех временных моментов множества $I_i^{K_i}$, $(r_j, \vartheta_j, \varphi_j, v_{r_j}, v_{\vartheta_j})^j$ - параметры найденного диполя для временных отсчетов интервала $[P_j - C_j]$, C_j - число временных отсчетов, начиная с P_j, для которых найденные параметры диполя не меняли значений.

В вышеописанных обозначениях сформулируем постановку задачи кластеризации. Путь дано множество примеров с определенными весами, $X^m = \{(r_m, \vartheta_m, \varphi_m, v_{r_m}, v_{\vartheta_m}, v_{\varphi_m})^m, w_m\}$, состоящее из всех элементов множества I^K. Требуется разбить множество на непересекающиеся подмножества - кластеры $K_1,...,K_l$, чтобы обеспечить минимум критерия функционала качества, т. е. найти такие

$$K = K_1,...,K_l : F(K) \to \min_K(\max),$$

$$F = \sum_{j=1}^{l}\sum_{i=1}^{n} \frac{w_{ij}^2}{n}, \quad (3)$$

где w_{ij} - степень принадлежности i-го объекта j-му кластеру, n -- число объектов, l -- число кластеров. В качестве меры близости 2--х примеров обучающего множества X было выбрано евклидово расстояние:

$$\forall\, x_i, x_j \in X^m, \rho_{ij} = \sqrt{(r_i - r_j)^2 + (\vartheta_i - \vartheta_j)^2 + (\varphi_i - \varphi_j)^2}.$$

Разработанный алгоритм кластеризации основывется на специфике задачи локализации. Весовые коэффициенты обучающих примеров множества фактически определяют количество временных отсчетов стационарного положения дипольного источника. В вышеописанных предположениях примеры из обучающего множества, имеющие наибольший весовой коэффициент являются активными с большой степенью достоверности, чем остальные. Исходя из построения весовых коэффициентов, упорядоченные множества I^K образуют последовательность рассмотрения возможных центроидов кластеров. Исходя из данных положений, предлагается рассматривать последовательность центроидов и определять число кластеров на основе минимизацией функционала качества. При этом центроид не может менять свои параметры, а только исключаться из рассмотрения.

Целью проведения кластерного анализа результатов локализации является как выделение наиболее достоверных зон активности, так и удаление из рассмотрения дипольных источников, являющихся

следствием артефактных событий (соотношением сигнал - шум). В рамках постановки задачи кластеризации определим нестационраные положения активности как множество таких примеров, у которых степень принадлежности к двум и более кластерам отличается на незначительную величину. Значение такой пороговой величины определяется экспериментально.

В вышеописанных предположениях сформулируем основные шаги алгоритма кластеризации источников активности:

1. Изначально множество цетроидов C пусто. В рассматриваемое множество центроидов добавляется первый элемент последовательности I^K (из I^K член последовательности исключается).
2. Для каждого примера i из обучающего множество вычисляется степень принадлежности j-му кластеру, т.е. соотвествующему центроиду из множества C: $w_{ij} = \left(\sum_{g=1}^{l} \left(\frac{\|x_i - v_j\|}{\|x_i - v_g\|} \right)^{2/m-1} \right)^{-1}$, где l -- число кластеров, $m > 1$, v_j -- параметры центроида j.
3. Каждому примеру ставится в соотвествие центроид (кластер) с наибольшим значением степени принадлежности $X^m = \{(r_m, \vartheta_m, \varphi_m, v_{r_m}, v_{\vartheta_m}, v_{\varphi_m})^m \to C_j\}$, где C_j - элемент множества C.
4. Вычисляется значение функции качества (3) для сформированного множества кластеров и критерий останова, основанный на сравнении изменения значения функции качества с пороговым значением (способ определения порогового значения описан в Приложении).
5. Если измнение функции соотвествует критерию останова, то алгоритм переходит к выполнению п.6, иначе шаги алгоритма повторяется с п.1.
6. Рассматриваются все примеры обучающего множества, для которых отличие степени принадлежности к двум различных кластерам минимально. Если весовой коэффициент таких примеров менее $15\tau_0$, т.е. соотвествует минимальному масштабу усреднения временных окон, то они исключаются из рассматриваемого множества.

7. Каждому кластеру ставится в соотвествие величина $|C_j| = \frac{|S_{C_j}|}{|S|}$ -- отношение числа примеров из исходного множества, для которых был найден кластер активности C_j, к общему числу примеров исходного множества, которая в последующем именутеся как мощность кластера.

3. Задача определения пространственных нейронных структур мозга, участвующих в обработке зрительной информации

Разработанный алгоритм построения усредненных карт активности был апробирован для исследования задач нейрофизиологии. В качестве входного сигнала использовались данные эксперимента. Был проведен атоматизированный анализ электроэнцефалограммы, построены усредненные карты активности для различных временных масштабов. Результаты апробации алгоритмов представлены в параграфе.

Проблема изучения нейрофизиогогических механизмов восприятия является одной из основных в нейрофизиологии. Человек воспринимает мир при помощи специализированных сенсорных систем - анализаторов. Восприятие - это процесс и результат формирования субъективного образа предмета или явления, действующего на анализатор. Интенсивное изучение нейрофизиологических механизмов восприятия стало возможным в связи с возникновением методов регистрации микро- и макропотенциалов мозга, т.е. активности отдельных нейронов и суммарной биоэлектрической активности мозга. При этом основной вопрос заключается в том, каким образом происходит прием и преобразование сенсорных стимулов и в каком виде отражается воспринятый и преобразованный стимул в центральной нервной системе человека.

В результате влияния внешних стимулов, действующих на человека в конкретной ситуации, в мозге человека формируется внутренне состояние, которое в последующем влияет на восприятие других стимулов. В психофизиологии такое явление имеет много названий, одно из них - установка [2]. После формирования такой установки при изменении внешнего стимула у испытуемого происходит несоответствие между внешним стимулом и внутренним образом (установкой), в результате имеют место ошибочные реакции. Обработка экспериментальных данных ЭЭГ позволяет изучать влияние

фиксированной установки на последующую мозговую деятельности человека.

Класс экспериментов, на которых был апробирован разработанный алгоритм построения карт активности, направлен на исследования неосознаваемых установок. Существует несколько методик экспериментов, связанных с выработкой установки. Одной из основных методик является предъявление человеку, участвующему в эксперименте (в дальнейшем именуемого как испытуемый) зрительных стимулов. В качестве важнейших внешних стимулов, предъявляемых во время эксперимента, нейрофизиологами были выбраны человеческие лица. Выражение человеческого лица имеет важное значение в межличностных отношениях. Оно служит одним из основных источников зрительной информации об эмоциональном состоянии и намерениях другого человека, содержит социальную и биологическую информацию, необходимую для организации адекватного поведения в конкретной ситуации [2]. С помощью анализа вызванных потенциалов мозга удалось показать, что восприятие эмоциональной экспрессии лица осуществляется при участии иерархически организованной разветвленной сети мозговых структур. Отдельные узлы этой структурно-функциональной системы активируются в определенной временной последовательности [4] что, по всей вероятности, отражает их последовательное участие в осуществлении отдельных этапов обработки зрительной информации. Однако, в настоящее время нет достаточных исследований, по изучению структур мозга активизирующихся при когнитивной деятельности, которая возникает при визуальном воспроиятии лиц, и их корреляции с другими отделами мозга. Разработанные программы определения активности позволяют получить новую информацию при изучении данных нейрофизиологических гипотез.

В работе предлагается использовать разработанные программы построения карт активности для анализа различных стадий эксперимента зрительного восприятия человеком стимулов и последующего их влияния на мозговую деятельность.

4. Протокол нейрофизиологического эксперимента, цели математической обработки экспериментальных данных

В рамках данного исследования была обработана электрическая активность испытуемых, принимавших участие в эксперименте, проводимом лабораторией Когнитивнх Процессов ИВНД и НФ РАН [3]. Описание и протокол эксперимента вместе с обработанными данными

были предоставлены лабораторией нейрофизиологии когнитивных процессов [4].

В экспериментах на здоровых взрослых людях исследовали влияние прошлого опыта испытуемых на функцию опознания эмоционального выражения лица. Сформулируем ряд определений для описания протокола эксперимента. Стадия формирования внутреннего состояния мозга называется стадией формирования установки, стадия действия установки на обработку мозгом внешних стимулов, называется стадией актуализации установки.

Протокол эксперимента. Иследовались 35 человек (20 женщин и 15 мужчин) в возрасте 25.1 ± 1.3 года.. Предварительно испытуемых знакомили в общих чертах с характером исследования. В центре монитора, находившегося на расстоянии 70 см от глаз испытуемого, на темно-сером фоне одновременно предъявлялись два кадра с изображением лица [5]. На стадии формирования установки слева предъявлялось сердитое выражение лица, справа - то же лицо с нейтральным, "спокойным" выражением. Время экспозиции -350 мс. После паузы 1 с в центре того же экрана появлялся пробный стимул - зеленое световое пятно диаметром 3 мм, с длительностью свечения 2 с. На стадии тестирования установки, которая начинается непосредственно после стадии формирования, одновременно предъявляли два кадра с нейтральным выражением того же лица с теми же параметрами стимуляции, что и на предыдущей стадии опыта. Испытуемый находился перед монитором компьютера в кресле в затемненной звукозаглушенной экранированной кабине, которая имела связь (микрофон-динамик) с помещением, в котором находился экспериментатор. Испытуемый получал инструкцию: "Сидите спокойно. Положите палец правой руки на кнопку. Смотрите на экран, будьте внимательны. На экране появятся изображения двух лиц, одно рядом с другим. Изображения исчезнут. Затем на экране появится световая точка. В ответ Вы должны как можно быстрее нажать на кнопку и после этого сообщить, одинаково ли выражение обоих лиц или же одно из них, по Вашему впечатлению, более неприятно. При этом используйте слова "одинаковые", если выражения лиц одинаковы, "слева", если более неприятно лицо слева, и "справа", если более неприятно лицо справа". На стадии формирования установки 15 раз последовательно предъявляли изображения лиц с разной эмоциональной экспрессией: слева - сердитое лицо, справа - нейтральное. На стадии тестирования установки во всех 30 пробах слева и справа предъявляли одно и то же лицо с нейтральным выражением. Экспертно множество из 45 показываемых стимулов было разделено на две стадии: стадию формирования установи с 1 по 15 стимулы, и стадию тестирования установки - с 16 по 45 стимулы.

Эксперимент разбивался на четыре стадии для сравнения электрических характеристик:

•стадия фоновой активности *"закрытые глаза"*, регистрация ЭЭГ сигнала у испытуемого перед началом эксперимента;

•стадия фоновой активности *"открытые глаза"* - стадия записи сигнала перед экспериментом, когда испытуемый открыл глаза;

•стадия *выработки установки*, т.е формирования неосознаваемого внутреннего состояния в виде установки.

•стадия *тестирования установки.*

На стадиях 3 и 4 в анализируемых массивах выделялись пред- и постстимульные фрагменты, которые соотвествуют временному промежтку до появления стимула на экране и сразу после исчезновения стимула.

Характеристики исследуемых экспериментальных данных .

Перед исследованием ЭЭГ сигналов проводилась предварительная обработка экспериментальных данных.

На первом этапе обработки записи ЭЭГ, сигнал фильтровался в диапазоне частот δ - β_2 ритмов. При этом отсекались высокоамплитудные колебания, связанные с движением глаз, напряжением мышц, и других артефактных явлений.

На втором этапе обработки вся запись эксперимента (17 минут) делилась на файлы, число которых равнялось числу предъявлений лицевых стимулов. Каждый файл начинался с метки начала предстимульной записи, которая менялась от 3 до 7 с, затем ставятся метки подачи стимула лица, метки подачи зеленой точки, метки нажатия на кнопку и 1 с постстимульной записи. Для каждого испытуемого имелось 45 файлов.

На третьем этапе из сигнала вырезались всплески, связанные с морганием глаз, имеющие определенные период возникновения и амплитуду. Если количество каналов с такими артефактами было больше чем 2/3 от общего числа каналов, то артефактный отрезок вырезался полностью.

Таким образом, было сформировано 15 файлов, относящиеся к периоду формирования установки и 30 - к тестированию. Средняя длина каждого из файлов составляла 5.5 секунд, из которых при помощи меток было выделено 4 секунды предстимульной записи и около 800 мс постстимульной записи. Частота дискретизации сигнала составляла 1000 Гц. В каждом из файлов выделялось 4000 временных моментов предстимульной записи и 800 временных моментов постимульной. Размер каждого файла составлял в среднем 1456 Кб. Общая длина записи проводимого эксперимента составляется 525 минут для 35 испытуемых, размерностью около 2.5 Гб.

Данные для диссертационных исследований были получены после двух описанных этапов обработки сигнала ЭЭГ.

Цели обработки экспериментальных данных.

Целью обработки описанных экспериментальных данных было исследование предложенного метода локализации источников и построения усредненных карт активности для его применения в качестве способа оценки функционального состояния мозговой активности. Данная апробация разработанных алгоритмов и программ может служить подтверждением применимости метода оценки для различных нейрофизиологических экспериментов, связанных с исследованием реакций человека на внешние стимулы. Полученные результаты локализации являются дополонительной информации об активности мозговых структур во время обработки внешних стимулов.

5. Результаты построения временных карт активности нейронных источников для различных стадий эксперимента

В работе представлены результаты обработки данных описанного эксперимента для двух испытуемых, в последующем обозначаемых как И1 и И2. Для каждого из испытуемых из сигнала ЭЭГ были выделены полосы альфа и тета ритмов, все стимулы эксперимента обрабатывались по двум множествам для предстимульной и постстимульной ЭЭГ: формирования установки и тестирования установки. В этом параграфе будут приведены результаты работы алгоритма обработки сигнала методом деревьев решений и построения кластеров вероятных областей активности для четырех стадий эксперимента и их сравнение: "закрытые глаза", "открытые глаза", предстимульная ЭЭГ, постстимульная ЭЭГ. Первичная стадия обработки сигнала включала в себя: удаление артефактов, фильтрация частот, фурье преобразование, получение файла для каждого стимула.

5.1 Выбор масштаба временного окна для построения пространственно-временных карт

Нахождение усредненной области положения источника для некоторого временного промежутка (временного окна) позволяет, с одной стороны, определить область стационарности процесса и, с другой, отследить изменения положения активного источника (множества источников), вызванные некоторым событием. Момент времени, когда произошло событие, по-видимому, соответствует смене функционального состояния мозга. Возникает вопрос о выборе

масштаба временного усреднения результатов локализации для потсроения карт активности.

Обозначим через τ_0 величину временного промежутка между временными отсчетами. В анализируемых экспериментах $\tau_0 = 10^{-3}$ сек. Временные промежутки для усреднения будем измерять в τ_0.

Рассмотрим масштабы временного окна для пространственно-временного усреднения локализованных источников, соответсвующие размерности стадий эксперимента: $\tau_1^1 = \tau_1^2 = 40000\tau_0$, $\tau_1^3 = 4000\tau_0$, $\tau_1^4 = 1600\tau_0$.Выбранный масштаб позволяет построить карты активности для стадий эксперимента, определенных экспертно, характеризуя среднее положение активных источников для всех событий, происходящих во время определенного фрагмента ЭЭГ. Обнаружение других смен функционального состояния мозговой активности при таком усреднении невозможно. Поэтому в работе рассматривается размер масштаба временного окна соответствующий среднему количеству временных точек, для которых временной процесс является стационарным. Экспериментально было получено приближенное среднее значение ``окна стационарности'' - $500\tau_0$ временных отсчетов, и в качестве масштаба для картирования активных зон было выбрано $\tau_2 = 500\tau_0$. Однако изменение вероятного положения дипольного источника может произойти на меньшем масштабе. Отношения сигнал/шум в исходных данных может являться следствием определения неверных зон локализации при уменьшении масштаба усреднения. В работе предлагается оценивать возможные отклонения от наиболее вероятных положений на основе выделения кластеров активности с определенными значениями мощности, которые позволяют получить взвешенную карту расположений дипольных источников. Из экспериментальных данных было рассчитано, что наименьшее стационарное положение дипольного источника в среднем по испытуемым равно 10 мс. Масштабирование с размером окна $\tau_3 = 10\tau_0$ отслеживает локальные области возникновения вероятных источников активности.

В результате проведенного анализа можно выделить три масштаба временного окна для получения усредненных оценок активности источников, основанные на первичной обработки результатов локализации входных данных ЭЭГ в каждой временной отсчет.

5.2 Построение пространственно-временных карт активности для фоновой ЭЭГ и при возникновении у испытуемого зрительно-иллюзорного образа

Фоновая запись сигнала ЭЭГ состоит из 2-х стадий: "закрытые глаза", "открытые глаза". Для каждой из стадий запись занимает 1 минуту (60000 τ_0 по 20 каналам), и после первичной обработки сигнала представляет 40 секунд отфильтрованного сигнала. Фурье преобразованием из сигнала выделяется альфа-активность и тета-активность, в среднем соответствующие диапазонам 7-13 Гц и 4-6 Гц. Характеристики входных данных представлена в Таблице 1. Величины приведенных характеристик совпадают для альфа- и тета-активности.

Таблица 1. Свободная таблица характеристик данных ЭЭГ альфа и тета - активности для четырех стадий эксперимента

Номер стадии эксрериме нта	Число временны х отсчетов	Число временных отсчетов после преобр. Фурье	Число временны х окон длиной 500 мс	Число временных окон длиной 10 мс
1,2	10^6	50000	100	5000
3,4	4000	5024	8	500

Таблица 2. Таблица размерности пространства локализованных зон альфа и тета - ритмов для четырех стадий эксперимента

Номер стадии экспериме нта	Число построенн ых деревьев	Число выделенных зон деревом до голосования	Число выделенных зон после голосования
1,2	50000	196000	50000
3,4	5000	25000	5000

После выделения временных окон для каждого из отсчетов строится дерево решений по алгоритму локализации дипольных исчтни ков, представленного в работе [1]. Для каждого временного отсчета выбирается одна область, и положение активного диполя в этой

области. Число построенных деревьев решений, и размерность пространства полученных решений представлена в таблице 2. Исходными данными для построения вероятностных карт активности для каждой их стадий являются 2 множества, для альфа- и тета-ритма (общий вид (2)):

$$I^{\alpha}_{40000} = (\{(r_{t_i}, \vartheta_{t_i}, \varphi_{t_i}, v_{r_{t_i}}, v_{\vartheta_{t_i}}, v_{\varphi_{t_i}})'\}, \{40000\});$$

$$I^{\vartheta}_{40000} = (\{(r_{t_i}, \vartheta_{t_i}, \varphi_{t_i}, v_{r_{t_i}}, v_{\vartheta_{t_i}}, v_{\varphi_{t_i}})'\}, \{40000\}),$$

где $t = 1..40000\tau_0$.

В данном случае они совпадают. Поскольку стимулов в состояниях фоновой активности не подавалось, то метка подачи стимула стоит на последнем временном отсчете. Для вычисления среднего положения диполя во время фоновой активности 40 секунд сигнала были разбиты на 10 подмножеств длиной 4 с, т.е. два исходных множества были разбиты на подмножества $I^{\vartheta} = I^{\alpha} = \{I_1, I_2, ..., I_{10}\}$,

$$I^{\vartheta}_i = I^{\alpha}_i = (\{(r_{t_i}, \vartheta_{t_i}, \varphi_{t_i}, v_{r_{t_i}}, v_{\vartheta_{t_i}}, v_{\varphi_{t_i}})'\}, \{4000\}),$$

$i = 1..10, t = 1..4000\tau_0$.

Разобьем все множества (полученные отрезки по 4 с) на временные окна длиной 500 мс для временного усреднения всех стимулов и 10 мс для анализа смещения и распространения активности при усреднении всех стимулов. Весовое значение было отнормированно и для положения каждого диполя равнялось 1. После построения упорядоченных последовательностей (3) для каждого значения длины окна

$$P^{\vartheta}_{1-10} = P^{\alpha}_{1-10}(\{I^{10}_1, I^{10}_2, ..., I^{10}_N\}^1;$$
$$\{I^{10}_1, I^{10}_2, ..., I^{10}_N\}^2; ; \{I^{10}_1, I^{10}_2, ..., I^{10}_N\}^{400})$$

были сформированы кластеры наиболее активных после усреднения зон.

Аналогично формируются последовательности для предстимульной и постстимульной стадии эксперимента. Подробное описание потсроения множеств для этих стадий представлено в работе [2].

Результатом обработки временных последовательностей будет множества карт активности для каждой из рассматриваемых стадий. В дальнейшем будем считать, что карту активности можно представить множеством параметров образующих ее кластеров.

Для сравнительного анализа различных стадий эксперимента, сравним параметры центроидов кластеров их образующих (Таблица 3).

Таблица 3. Множества центров кластеров вероятных положений источников по всем стадиям эксперимента

Стадия ``закрытые глаза"	Стадия ``открытые глаза"	Стадия предстимульной ЭЭГ	Стадия постстимульной ЭЭГ
Наиболее вероятное положение дипольного источника альфа и тета ритмов			
$K_1 =$ $\{x_p = -0.0683,$ $y_p = 0.2904,$ $z_p = 0.4742\}$	$K_1 =$ $\{x_p = -0.0683,$ $y_p = 0.2904,$ $z_p = 0.4742\}$	$K_1 =$ $\{x_p = -0.0683,$ $y_p = 0.2904,$ $z_p = 0.4742\}$ $K_3 =$ $\{x_p = 0.2176,$ $y_p = 0.2044,$ $z_p = 0.2398\}$	$K_6 =$ $\{x_p = 0.3305,$ $y_p = 0.0124,$ $z_p = 0.1008$ $K_7 =$ $\{x_p = 0.689,$ $y_p = -0.0117,$ $z_p = 0.2475\}$
Вероятные положения дипольного источника альфа и тета ритмов			
	$K_2 =$ $\{x_p = -0.0336,$ $y_p = 0.0993,$ $z_p = 0.3605\}$	$K_4 =$ $\{x_p = 0.1293,$ $y_p = -0.161,$ $z_p = 0.3447\}$ $K_2 =$ $\{x_p = -0.0336,$ $y_p = 0.0993,$ $z_p = 0.3605\}$ $K_5 =$ $\{x_p = 0.2716,$ $y_p = -0.2044,$ $z_p = 0.2398\}$	$K_8 =$ $\{x_p = -0.0557,$ $y_p = -0.0794,$ $z_p = 0.2159\}$

В таблице 3 показаны множества центров кластеров, соответсвующие вероятным положениям источников по всем стадиям эксперимента. Сравнение строчек таблицы позволяет различить все фрагменты сигнлала ЭЭГ на основе построенных множеств кластеров. Множества наиболее вероятных положений дипольных источник для стадий: ``закрытые глаза", ``открытые глаза", предстимульного фрагмента ЭЭГ пересекаются в позиции K_1. Данное пересечение объясняется тем, что три стадии относятся к фоновому сигналу, когда

нет внешних стимулов и испытуемые постепенно меняют свое состояние, открывая глаза и готовясь к зрительному восприятию. При сравнении первых трех множеств с множеством постстимульного фрагмента ЭЭГ общих позиций не найдено, это говорит о том, что источник активности переместился после восприятия зрительного стимула. Пересечение, объединение множеств возможных позиций определяется возможные различия в стадиях и определяет те фрагменты, которые не имеют общих точек, значит, построение таких множеств может являться оценкой функционального состояния мозговой деятельности.

6. Результаты

В работе представлен алгоритм построения карт активности для получения усредненных оценок положения активных зон группы сигналов ЭЭГ. Результаты применения алгоритма для обработки экспериментальных данных нейрофизиологических исследований показали, что данная методика может служить новым методом оценки функционального состояния мозговой активности.

Список литературы

1. *Анализ электрической активности человечкского мозга на основе ансамблей деревьев решений.* **Е.А., Попова.** 2000, М.: Вестн.Моск.Ун-та Сер. 15 Вычисл. матем. и кибер. С. 46-55.

2. Костандов ЭЛ., Курова Н.С., Черемушкин ЕЛ., Яковенко ИЛ. *Зависимость установки от участия вентральной и дорзальной зрительных систем в когнитивной деятельности.* Журн. высш. нерв. деят. 2005. Т. 55. № 2. С. 156-163.

3. Лаборатория нейрофизиологии когнитивных процессов Института Нейрофизиологии и Высшей нервной деятельности и РАН *http://www.ihna.ru/book.php?=1\&id=22*

4. Михайлова Е.С. *Нейробиологические основы опознания человеком эмоций по лицевой экспрессии.* Журн. высш. нерв. деят. 2005. Т. 55. № 1. С. 15-28.

5. Костандов ЭЛ., Курова Н.С., Черемушкин Е.А., Яковенко ИЛ.,Петренко Н.Е., Ашкинази М.Л. *Установка как регулирующий фактор в функции опознания эмоционального выражения лица.* Журн. высш. нерв. деят. 2006. Т. 56. № 5. С. 581-589.

Раздел III

Обработка текстовой информации

Гудков А.В.

Разработка интерпретатора Лисп-подобного языка для исследования алгоритмов распределенного поиска.

Введение

Поисковые системы – один из инструментов, известный каждому пользователю сети Интернет. Однако быстрый рост объема информации требует от поисковых систем постоянного улучшения используемых алгоритмов, большинство из которых допускает параллельную работу на вычислительном кластере. На текущий день известен ряд алгоритмов поиска, показавших свою эффективность. Использование комбинации таких алгоритмов позволяет устранить недостатки каждого алгоритма в отдельности.

Для исследования таких алгоритмов предлагается универсальный язык, позволяющий использовать как основные существующие алгоритмы, так и их комбинации.

Выбор языка запросов

Язык запросов (ЯЗ) должен удовлетворять следующим требованиям:

1. иметь встроенный набор операций, достаточный для проведения исследования;
2. возможность интерпретирования языка в режиме реального времени, без перекомпиляции программы поисковой системы;
3. возможность описывать параллельное выполнение программных конструкциях, так как большинство алгоритмов допускает распараллеливание;
4. возможность лаконично описать запрос;
5. иметь возможность расширения за счет введения новых операторов.

Лисп – один из языков, близких к указанным требованиям. Однако, требуется его расширение для удовлетворения всех требований.

Описание поисковой модели

Поисковая модель предполагает наличие набора консистентных по отдельности, но логически связанных сегментов, построенных на основе исходной коллекции данных. Каждый сегмент содержит в себе следующие структуры: прямой индекс (или, другими словами, репозитарий) – предоставляет по указанному уникальному номеру гипертекстовой страницы производить выборку содержимого этой страницы; обратный индекс – позволяет по указанному слову производить выборку списка номеров страниц, в тексте которых содержится указанное слово.

Логическая связанность означает, что сегменты неким образом покрывают исходную коллекцию страниц, и поиск внутри всей коллекции требует использования алгоритма, который будет обращаться к нескольким (или даже ко всем) сегментам.

Рассмотрим три вида логической связанности и соответствующие алгоритмы:

1. один сегмент S – вся коллекция индексируется в один сегмент, расположенный на одном вычислительном узле;
2. сегменты $S_1, S_2, \square, S_n$, разбитые по документам [1, стр. 61-64]. В этом случае исходная коллекция C делится на коллекции $C_1, C_2, \square, C_n$ из непересекающихся множеств страниц. Затем на основе каждой коллекции C_i строится сегмент S_i. Каждый из сегментов располагается на своем вычислительном узле. Алгоритм на первой стадии делает запросы ко всем сегментам, затем объединяет результаты. Первая стадия может выполняться параллельно;
3. сегменты, разбитые по словам [1, стр. 389-390]. Исходная коллекция C, которая покрывает множество слов $\{w_1, w_2, \square, w_k\}$, делится на следующие коллекции: C_1 со словами $\{w_1^1, w_2^1, \square, w_{k_1}^1\}$, C_2 со словами $\{w_1^2, w_2^2, \square, w_{k_2}^2\}$ и т.д., где $w_{i_1}^{j_1} \neq w_{i_2}^{j_2}$ *при* $i_1 \neq i_2$, $j_1 \neq j_2$. Каждый из сегментов располагается на отдельном вычислительном узле. Алгоритм сначала получает списки требуемых слов (параллельно), затем строит их пересечение.

На выходе каждого из алгоритмов образуется список номеров страниц, которые удовлетворяют исходному запросу. Далее эта информация выводится пользователю поисковой системы, возможно, с некоторыми дополнительными атрибутами, полученными из прямого индекса (например, отрывок из текста страницы, содержащий искомые слова).

Расширение набора стандартных процедур языка

Операторы языка запросов делятся на две категории: стандартные (заимствованные из операторов языка Лисп) и расширенные (работающие в терминологии вышеописанной поисковой модели).

К расширенным оператором относятся: операторы работы с сегментами seg_bind, seg_open, seg_load, seg_intersect, seg_union, seg_minus и оператор параллельного выполнения вычислений – par_eval.

Оператор seg_bind подготавливает сегмент для работы и делает его активным. Модель допускает наличие только одного активного сегмента на каждом вычислительном узле в каждый момент времени. Функция принимает один аргумент – путь в файловой системе, где расположен сегмент. Пример вызова: (seg_bind «/home/user/segment»).

Оператор seg_open принимает один аргумент – слово. Возвращает объект-итератор по номерам документов, находящийся на диске, в тексте которых содержится искомое слово. Требует наличия активного сегмента. Пример вызова: (seg_open «автомобиль»).

Оператор seg_load принимает один аргумент – объект-итератор по номерам документов. Загружает номера документы в оперативную память и возвращает итератор по ним. Введение этого оператора обусловлено тем, что одновременное использование двух итераторов по номерам документов, находящимся на диске, часто оказывается неэффективным за счет большого количества операций ввода/вывода. Пример вызова: (seg_load (seg_open «автомобиль»)).

Оператор seg_intersect принимает два аргумента – два объекта-итератора по номерам документов. Возвращает итератор по пересечению номеров документов, загруженных в память. Аргументы вычисляются слева направо. Пример вызова: (seg_intersect (seg_load (seg_open «автомобиль»)) (seg_open «страховка»)).

Операторы seg_union и seg_minus работают аналогичным образом и представляют собой объединение и дополнение соответсвенно.

Оператор par_eval имеет следующую структуру: (par_eval адрес$_1$ тело$_1$ адрес$_2$ тело$_2$ .... адрес$_n$ тело$_n$), где адрес$_i$ – сетевой адрес вычислительного узла, где должно быть выполнено тело$_i$. Оператор выполняет вычисления параллельно, пересылая все тело$_i$ на адрес$_i$, и

собирая результаты в список (результат$_1$ результат$_2$ ... результат$_n$). Таким образом, par_eval имеет модель работы scatter/gather.

Заметим, что понятие «объект-итератор» относится к некоторой структуре языка программирования (реализующий язык - РЯ), на котором строится интерпретатор ЯЗ. Все введенные операторы также вызывают некоторые функции РЯ.

Примеры запросов

Теперь покажем, как с помощью предлагаемого ЯЗ построить запросы для всех трех алгоритмов. Для этого будем рассматривать запрос «покупка продажа авто».

Алгоритм №1. Сегмент расположен на вычислительном узле с сетевым адресом $id1$. Запрос строится следующим образом:

```
(par_eval
  id1 (quote (seg_intersect
                  (seg_load (seg_open «покупка»))
                  (seg_intersect
                        (seg_load (open «продажа»))
                        (seg_open «авто»)))))
```

Алгоритм №2. Даны два сегмента, содержащие по 50% исходной коллекции C, с сетевыми адресами $id1$ и $id2$ соответсвенно. Запрос строится следующим образом:

```
(define query (quote
      (seg_intersect
            (seg_load (open «покупка»))
            (seg_intersect
                  (seg_load (open «продажа»))
                  (seg_open «авто»)))))
(seg_union
      (par_eval id1 query id2 query))
```

Алгоритм №3. Даны два сегмента с разбиением по словам, функция where возвращает сетевой адрес по слову, где расположен сегмент, в покрытие которого входит это слово. Запрос строится следующим образом:

```
(define join_all (lambda (x)
      (cond
            ((null (cdr x)) (car x))
            (T (intersect (car x) (joinall (cdr x)))))))
(join_all (par_eval
      (where "покупка") (quote (seg_load (seg_open "покупка")))
```

```
(where "продажа") (quote (seg_load (seg_open "продажа")))
(where "авто") (quote (seg_load (seg_open "авто")))))
```

Обобщенная функция поиска для первого алгоритма:

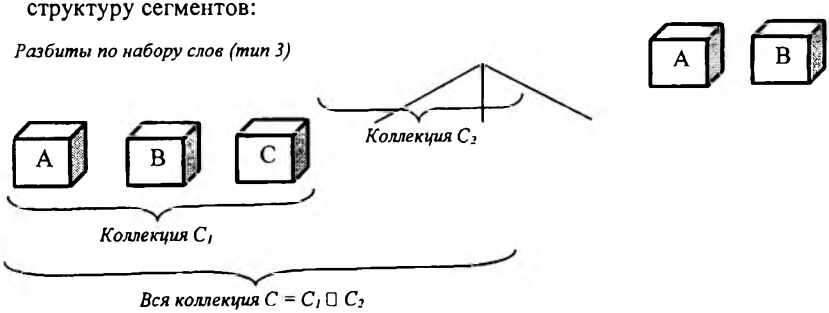
```
(define get (lambda (w)
        (seg_load (seg_open w))))

(define load_all (lambda (x)
        (cond
                ((null x) nil)
                 (T (cons (get (car x)) (load_all (cdr x)))))))

        (define search1 (lambda (x)
                (join_all (load_all x))))
```

Обобщенные функции поиска для второго и третьего алгоритмов строятся аналогично – на первом этапе происходит загрузка страниц и передача на один вычислительный узел, на второй стадии – поиск пересечения полученных списков страниц. Для работы третьего алгоритма требуется ввести функцию where, возвращающую адрес вычислительного узла, где расположен сегмент, в покрытие которого входит слово-аргумент функции. Обычно разбиение множества слов по сегментам производится с помощью подобранной хэш-функции от слова. Такой подход имеет ряд полезных свойств: простота вычисления и автоматическая балансировка размеров сегментов.

Полученные результаты

Реализация интерпретатора ЯЗ была создана на языке Java. Реализация включает в себя адаптер для работы с поисковой системой, собственно, сам интерпретатор, и эмулятор кластера. На основе тестовой коллекции были созданы необходимые сегменты.

Реализация позволяет исследовать работу алгоритмов поиска и их комбинаций. Как пример, рассмотрим следующую иерархическую структуру сегментов:



Такой подход удобно применять в поисковых системах, работающих с постоянно меняющимися данными, например, с новостными сайтами [1, стр.64-66]. В этом случае коллекция C_1 представляет из себя основной сегмент, а C_2 – постоянно пересоздаваемый сегмент за счет обновляемых данных. Так как размер C_2 мал по сравнению с C_1, то операция пересоздания требует малого количества времени. Когда размер C_2 превысит допустимый верхний порог, запускается механизм объединения сегментов, и новый сегмент становится на место C_1, а C_2 – опустошается.

Дополнительные свойства рассмотренного ЯЗ:

1. Производительность ЯЗ. ЯЗ используется исключительно для высокоуровнего контроля вычислений; операции пересечения, объединения и дополнения, требовательные к системным ресурсам, создаются на РЯ. Поэтому общая производительность в основном зависит от выбора РЯ.

2. Переносимость. ЯЗ может быть использован для любых поисковых систем, удовлетворяющих описанной модели. При этом конструкции запросов на ЯЗ не меняются, но требуется написание введенных функций на РЯ в соответствии с интерфейсом используемой поисковой системы.

Заключение

Было показано, что представленный ЯЗ может быть использован для реализации распределенных алгоритмов поиска, ровно как и для реализации их суперпозиции.

Литература

1. Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze. An Introduction to Information Retrieval. Preliminary Draft. - Cambridge UP, 2007

2. Ian H. Witten, Alistair Moffat, Timothy C. Bell. Managing Gigabytes. Compressing and Indexing Documents and Images. - Morgan Kaufmann, 1999

3. Richard Kelsey, William Clinger, Jonathan Ress. Report on the Algorithmic Language Scheme - 2007

Мальковский М.Г., Старостин А.С.

Синтаксический анализатор Treevial. Постановка задачи синтаксического анализа

Введение

Анализатор Treevial – это синтаксический анализатор текстов на естественном языке. Он является частью системы поддержки проектирования лингвистических процессоров Treeton, которая разрабатывается на кафедре алгоритмических языков факультета ВМиК МГУ, начиная с 2005 года.. В основе анализатора Treevial лежит принцип восходящего синтаксического анализа, в ходе которого порождаемые структуры оцениваются с помощью аппарата штрафных функций. Штрафные функции нужны для того, чтобы отличать правильные гипотезы о синтаксическом строении предложения от неправильных. Чем больше значение штрафной функции для определенной структуры, тем менее правдоподобной она считается. Оценка структур производится динамически и постоянно корректирует направление переборного процесса. Для задания штрафных функций могут использоваться любые языковые явления, которые можно формально исчислить. Авторы часто опираются на различные явления, связанные с топологией синтаксических структур. К примеру, известно, что предложения естественного языка в массе своей проективны. Поэтому, даже такая простая штрафная функция, которая ставит в соответствие всем проективным предложениям 0, а непроективным 1, может быть успешно использована на практике. Реальные штрафные функции, учитывающие топологию синтаксических структур, описаны в [2].

Язык описания синтаксических правил является декларативным. Каждое правило утверждает допустимость некоторой конфигурации в синтаксических структурах описываемого естественного языка (например, правило может декларировать, что две единицы, относящиеся к тем или иным классам, могут связываться определенной синтаксической связью или объединяться в группу). Синтаксис языка описания правил в этой статье не рассматривается[1]. Стоит лишь сказать, что авторы старались сделать язык достаточно универсальным, чтобы охватить как можно больший круг лингвистов – потенциальных пользователей анализатора. Он в определенном смысле согласован как с грамматиками зависимостей [5], так и с грамматиками

[1] Описание синтаксиса языка правил и примеры синтаксических правил можно найти в [4]

непосредственных составляющих [6]. Возможен также смешанный подход (по аналогии с [2]).

В данной работе приводится описание математической модели анализатора Treevial. Авторы стремились сосредоточиться на описании сути вводимых понятий, сознательно абстрагируясь от конкретных способов записи синтаксических правил, штрафных функций и т.п. В первом разделе вводятся понятия синтаксической структуры, мультиструктуры и синтаксического правила. Далее формально ставится задача синтаксического анализа. Кроме того, приводится расширенная постановка задачи синтаксического анализа, учитывающая штрафные функции.

Основным мотивом для создания данной статьи было желание заново структурировать и осмыслить плоды продолжительной и интенсивной работы над анализатором. Кроме того, по мнению авторов, язык математики как нельзя лучше подходит для описания принципов работы сложной системы, т.к. позволяет полностью абстрагироваться от тонкостей реализации. Эта статья адресована, прежде всего, математикам, которых интересуют проблемы моделирования естественного языка. Мы надеемся, что представленная в статье модель вызовет интерес со стороны исследователей такого рода.

1. Синтаксические структуры и правила

Прежде, чем приступить к формальному изложению, перечислим базовые понятия, с которыми работает анализатор Treevial, и приведем краткие пояснения:

- Наборы свойств и пространство категорий

В системе Treeton информация о входном тексте хранится в виде набора древовидных структур, называемых тринотациями (рабочий термин, родившийся из соединения англ. "tree" и "annotation"). Синтаксические структуры, порождаемые анализатором Treevial, также представляются тринотациями. Понятие тринотации расширяет часто используемое в системах обработки текстов понятие аннотации – набора пар вида <атрибут, значение>, соотнесенного с отрезком текста (ср. [1]). Для краткости будем называть такой набор набором свойств.

Наборы свойств используются для того, чтобы соотносить с тринотациями определенные лингвистические характеристики. Так, например, наличие в наборе свойств пары $<POS=N>$ обозначает то, что данная тринотация маркирует существительное, а наличие пары $<CAS=acc>$ – то, что тринотация маркирует некоторую единицу в винительном падеже.

Если перечислить все допустимые атрибуты и их возможные значения, то становится возможным построить множество, в которое входят все наборы свойств, которые в принципе могут возникнуть в процессе анализа. Мы будем называть такое множество пространством категорий.

- Словоформа

Под словоформой понимается графическая запись некоторого слова естественного языка, с которой соотнесен набор свойств, описывающий грамматическую информацию[1]. Например, для существительного мужского рода «стол» словоформа творительного падежа выглядит так:

столом

<base=стол, POS=N, GEND=m, ANIM=inan, NMB=sg, CAS=inst>

- Словосочетание

Под словосочетанием понимается упорядоченный набор словоформ. Ниже словосочетания будут использоваться как для представления анализируемых предложений целиком, так и для представления отдельных частей предложений. Так, словосочетанием является предложение «мама мыла раму мылом» при условии, что заранее известны морфологические характеристики всех слов (например про слово «мыла» известно, что это глагол, а не форма родительного падежа слова «мыло» и т.п.). Часть приведенного предложения (например, «мыла мылом») также может рассматриваться как отдельное словосочетание.

- Синтаксическая структура

В лингвистике существует представление о том, что предложения естественного языка имеют неявно выраженную внутреннюю структуру. Считается, что в любом предложении слова некоторым образом связаны между собой или организованы в группы. Причем, большинство исследователей сходятся в том, что эта структура изоморфна дереву. Существует два основных подхода к описанию синтаксических структур: деревья зависимостей (ср. [5]) и системы составляющих (ср. [6]). В рамках первого подхода утверждается, что структура предложения представляет собой дерево с размеченными дугами, узлами которого являются слова. В рамках второго подхода

---

[2] С первого взгляда может показаться, что авторы понимают словоформу не совсем так как это принято в современной науке. На самом деле, соотнося с графической записью набор свойств, мы, наоборот, подчеркиваем то, что у словоформы есть две стороны: внешняя и внутренняя. Причем во внутреннюю традиционно принято включать синтаксические характеристики (см. [3]).

утверждается, что слова в предложении организованы в систему вложенных друг в друга групп (составляющих), причем каждая группа может быть непосредственно вложена не более чем в одну группу. В работе [2] А.В. Гладкий предложил подход к описанию синтаксических структур с помощью систем синтаксических групп (ССГ), объединяющий достоинства обоих упомянутых подходов. Авторы приняли решение использовать внутри анализатора Treevial структуру аналогичную ССГ. Под синтаксической структурой понимается дерево с размеченными дугами, причем одна метка (w) имеет специальное назначение. Ниже иллюстрируется то, каким образом вводимое понятие обобщает деревья зависимостей и системы составляющих.

- Синтаксическое правило

Синтаксические правила являются основным инструментом для описания того, какие синтаксические структуры допустимы в описываемом естественном языке. Каждое синтаксическое правило описывает некоторый класс ситуаций, которые считаются допустимыми. По аналогии с синтаксическими структурами правила представляют собой деревья с размеченными дугами. Только в узлах этих деревьев стоят уже не конкретные словоформы, а шаблоны, т.е. формальные описания некоторых классов словоформ (например, все существительные в винительном падеже и т.п.). Кроме того, дуги и узлы синтаксических правил делятся на исходные и производные. Это позволяет контролировать порядок применения правил. На текущем шаге переборного процесса сопоставление происходит только с исходными элементами правила, после чего к структуре добавляются дуги и узлы, соответствующие производным элементам. Еще одним важным элементом синтаксического правила является общее ограничение на все шаблоны правила. С помощью этого ограничения становится возможным учитывать зависимости между свойствами различных узлов. Например, согласование по той или иной категории (совпадение падежей и т.п.). Таким образом, каждое правило неявно описывает класс древесных фрагментов, которые могут встречаться в синтаксических структурах моделируемого языка. Следует отметить, что обычно деревья синтаксических правил состоят из малого количества узлов (редко их бывает больше 4). Самый частый вид синтаксического правила – это бинарное правило, в котором два шаблона связаны одной синтаксической связью. Приведем пример такого правила для русского языка:

```
A(POS=A) –modif-> B(POS=N) ::
A.CAS == B.CAS && A.NMB == B.NMB &&
A.ANIM == B.ANIM && A.GEND == B.GEND
```

Это правило утверждает, что в синтаксических структурах предложений русского языка могут встречаться фрагменты, в которых существительные связаны связью modif с прилагательными, согласующимися с ними по роду числу и падежу. В этом правиле элементы A и B являются исходными, а дуга, помеченная *modif*, является производной.

- Мультиструктура

Для заданного словосочетания набор синтаксических правил неявно определяет процесс «сборки» синтаксической структуры, на каждом шаге которого применяется то или иное правило. Заметим, что хотя целевая структура по определению древесна, на всех промежуточных стадиях этого процесса будет фигурировать не дерево, а лес. Этот лес авторы называют мультиструктурой. Следует отметить, что исходное словосочетание также является мультиструктурой, т.к. набор разрозненных точек является лесом. Таким образом, процесс сборки синтаксической структуры представляет собой последовательность преобразований мультиструктуры – от несвязного набора точек-словоформ к связному дереву. В процессе последовательного преобразования мультиструктуры могут образовываться синтаксические структуры, покрывающие все входное словосочетание. Результатом синтаксического анализа считается набор всех таких структур.

Ниже приводится формальное описание всех упомянутых понятий, описывается процедура применения правила к мультиструктуре и формально ставится задача синтаксического анализа.

Будем считать, что все атрибуты составляют конечное множество *ATTR*, а все их возможные значения заданы в виде инъективного отображения $VAL: ATTR \rightarrow \{Y_i | Y_i$ – конечное или счетное множество$\}$.

Определение 1.1. *Пространством категорий C* называется множество вида $\{((f_1, v_1), (f_2, v_2), ..., (f_n, v_n)) \mid 0 \le n, f_i \in ATTR, v_i \in VAL(f_i)$ для всех $1 \le i,j \le n, f_i \ne f_j$ для i ≠ j$\}$

Понятие пространства категорий потребуется ниже при описании математической модели анализатора Treevial. Следует отметить, что пространство категорий является аналогом вводимого в рамках классической теории грамматик понятия множества терминалов.

Будем считать, что все синтаксические связи, которые необходимо различать, образуют конечное множество *RELS*. Словоформу будем представлять как точку на оси целых неотрицательных чисел, с которой соотнесен определенный элемент пространства категорий.

Определение 1.2. Будем называть *словосочетанием* пару *(M,f)*, где

$M=m_1,\ldots,m_n$ – упорядоченный набор словоформ

f – функция категориальной разметки: f: $M \rightarrow C$

Определение 1.3. *Синтаксической структурой S(p) словосочетания* $p=(M,f)$ будем называть набор $(T(V=M\cup V',E),g,h)$, где

$T(V,E)$ – направленное дерево (V – множество вершин, E – множество упорядоченных пар (a,b), $a,b \in V$)

V' – множество виртуальных узлов

g – функция разметки вершин: g: $V' \rightarrow C$, причем для всех $m \in M$, $g(m)=f(m)$

h – функция разметки дуг: h: $E \rightarrow RELS \cup \{w\}$, w – служебная связь, символизирующая вложение (см. ниже), причем для всех $(m,y) \in E$, $m,y \in M$, $h((m,y))\neq w$[1]

Синтаксическая структура представляет собой дерево с размеченными узлами и дугами, «надстроенное» над словосочетанием, – в него входят все словоформы словосочетания а также набор виртуальных[2] узлов. Будем говорить также, что синтаксическая структура *опирается* на словосочетание.

В нашей модели каждая словоформа, принадлежащая словосочетанию, однозначно привязана к числовой оси своим порядковым номером. Тем самым отражается тот факт, что словосочетания естественного языка развертываются в одном направлении и слова в них не накладываются друг на друга. Широкий класс виртуальных узлов также может быть привязан к числовой оси. Для этого используется служебная связь w, с помощью которой задаются функции $s(v)$, $e(v)$, определяющие точки начала и конца для узлов синтаксической структуры[3]:

---

[3] Последнее условие означает, что словоформы в нашей модели атомарны – они не могут иметь внутреннюю структуру.

[4] Термин виртуальный в данном случае употребляется, чтобы подчеркнуть тот факт, что, в отличие от словоформ, эти узлы не имеют акустического или графического выражения. В этом смысле дуги синтаксической структуры также являются виртуальными объектами.

[5] При этом будем использовать специальное обозначение *null* для неопределенной величины. Считается, что функции max и min игнорируют значения *null*:

$$\max(null, v_1, v_2, \ldots, v_n) = \max(v_1, v_2, \ldots, v_n); \max(null) = null;$$

$$\min(null, v_1, v_2, \ldots, v_n) = \min(v_1, v_2, \ldots, v_n); \min(null) = null;$$

$$s(v)\begin{cases}\min s(v_j),\ (v,v_j)\in E, g((v,v_j))=w\\ i, \text{когда } v=v_i\in M\\ null, \text{если } v\notin M \text{ и } \forall(v,x)\in E, g((v,x))\neq w\end{cases}$$

(1.1)

$$e(v)\begin{cases}\max e(v_j),\ (v,v_j)\in E, g((v,v_j))=w\\ i, \text{когда } v=v_i\in M\\ null, \text{если } v\notin M \text{ и } \forall(v,x)\in E, g((v,x))\neq w\end{cases}$$

(1.2)

Легко видеть, что при таком определении функций *s* и *e*, наличие связи *w* между узлами символизирует то, что один узел становится как бы «оболочкой» поддерева, корнем которого является другой узел.

Очевидно, что для словоформы значением обеих функций всегда будет ее привязка к числовой оси. Остальные же узлы относительно введенных функций можно разделить на два класса. В первый класс войдут все узлы, для которых $s(v)\neq null$ и $e(v)\neq null$, а во второй все узлы, для которых $s(v)=e(v)=null$. Узлы, относящиеся к первому классу, мы будем называть *составляющими*, а узлы, относящиеся ко второму классу, – *связками*.

Описанная выше модель синтаксической структуры оказывается достаточно универсальной и допускает различные подходы к описанию поверхностного синтаксиса. Так, те лингвисты, которые используют деревья зависимостей, могут работать с описанной моделью, не используя составляющих, а лингвисты, использующие системы составляющих, могут, наоборот, использовать только составляющие и не использовать именованные связи.

Прежде чем перейти к описанию синтаксических правил, приведем определение мультиструктуры.

Определение 1.4. *Мультиструктурой $S^m(p)$ словосочетания $p=(M,f)$* будем называть набор $(W(V=M\cup V',E),g,h,<\Omega_1,V_1>\ldots<\Omega_m,V_n>)$, где

$W(V,E)$ – направленный лес (V – множество вершин, E – множество упорядоченных пар (a,b), $a,b\in V$)

V' – множество виртуальных узлов

g – функция разметки вершин: g: $V'\rightarrow C$, причем для всех $m\in M$, $g(m)=f(m)$

h – функция разметки дуг: h: $E\rightarrow RELS\cup\{w\}$, причем для всех $(m,y)\in E$, $m,y\in M$, $h((m,y))\neq w$

V_j – упорядоченный набор узлов, причем $V_j \subset V$,

$\Omega_j \in P(\{(x,y) \mid x,y \in N_0 \cup \{null\}\}^{|Vj|})$.

Набор $<\Omega_j, V_j>$ определяет последовательность многомерных ограничений на функции $s(v)$ и $e(v)$, сформированную в процессе применения синтаксических правил. Его назначение прояснится ниже.

Мультиструктуру, соответствующую исходному словосочетанию, мы будем называть *исходной мультиструктурой.*

Определение 1.5. *Исходной мультиструктурой словосочетания* $p=(M,f)$ называется мультиструктура вида $(W(V=M,\varnothing),f,\varnothing,\varnothing)$.

Анализатор работает с конечным множеством правил, каждое из которых является описанием класса ситуаций, при которых мультиструктуры определенного вида могут быть определенным образом преобразованы.

Каждое синтаксическое правило представляет собой дерево, узлы и дуги которого делятся на исходные и производные. При применении синтаксического правила r к мультиструктуре S^m исходные узлы и дуги r сопоставляются с узлами и дугами S^m, после чего к S^m добавляются узлы и дуги, соответствующие производным узлам и дугам r.

Приведем формальное определение синтаксического правила и поясним его:

Определение 1.6. *Синтаксическим правилом* называется набор $(T(V_o \cup V_n, E_o \cup E_n), \alpha_o, \alpha_n, \beta, \Psi)$, где

$T(V_o \cup V_n, E_o \cup E_n)$ – направленное дерево. V_o – упорядоченный набор исходных вершин, V_n – множество производных вершин. E_o – множество исходных дуг, E_n – множество производных дуг. $V_o \neq \varnothing$. $E_n \neq \varnothing$. Для $\forall (v_1,v_2) \in E_o$: $v_1 \in V_o$, $v_2 \in V_o$

α_o – функция разметки исходных вершин: α_o: $V_o \rightarrow P(C)$

α_n – функция разметки производных вершин: α_n: $V_n \rightarrow \{f$: $C^{|Vo|} \rightarrow C\}$

β – функция разметки дуг: β: $E_o \cup E_n \rightarrow RELS \cup \{w\}$.

Ψ – область многомерных ограничений: $\Psi \in P(C^{|Vo|} \times \{(x,y) \mid x,y \in N_0 \cup \{null\}\}^{|Vo|})$

Функция α_o соотносит с каждой исходной вершиной правила некоторую область пространства категорий. Эту область можно также называть одномерным шаблоном. Сопоставление некоторого узла мультиструктуры с узлом правила становится возможным только тогда, когда элемент пространства категорий, соответствующий узлу мультиструктуры, попадает в область, соответствующую узлу правила.

Функция α_n соотносит с каждой производной вершиной правила некоторую функцию, позволяющую определить элемент пространства категорий по набору из $|Vo|$ элементов пространства категорий. Когда

применяется правило, эта функция требуется для соотнесения некоторого элемента пространства категорий с только что добавленным узлом мультиструктуры. Аргументом функции становится набор из $|Vo|$ элементов пространства категорий, соответствующих узлам, сопоставленным исходным вершинам правила.

Функция β сопоставляет как исходным, так и производным дугам правила фиксированные элементы множества возможных связей (здесь и далее будем называть так множество $RELS \cup \{w\}$). Для исходных дуг этот элемент используется в момент применения правила как шаблон для проверки, а для производных дуг, наоборот, как константная функция.

Область Ψ используется для того, чтобы задавать те ограничения на применение правила, в которых участвует более одного узла (например, согласование нескольких элементов по какой-то категории). Для проверки применимости правила составляется набор из элементов пространства категорий, соотнесенных с узлами, сопоставленными с исходными вершинами правила, и из значений функций $s(v)$ и $e(v)$, для этих узлов. Для того, чтобы правило было применимо, необходимо, чтобы построенный набор попадал в область Ψ.

Правило может быть применено к мультиструктуре только тогда, когда узлы мультиструктуры можно сопоставить исходным вершинам правила так, что элементы пространства категорий, соответствующие узлам мультиструктуры, попадут в области, соответствующие вершинам правила. Кроме того, поскольку такое сопоставление определяет для каждой исходной дуги правила упорядоченную пару узлов, принадлежащих мультиструктуре[1], необходимым условием применимости является также и то, чтобы каждую такую пару узлов в мультиструктуре соединяла дуга, помеченная той же связью, что и дуга правила. Другими словами, элементы мультиструктуры должны сопоставляться с узлами правил вместе с соединяющими их дугами.

Еще одним важным условием применения правила является то, что после добавления к мультиструктуре новых узлов и дуг полученный граф должен оставаться лесом. Это условие означает, что новые дуги не добавляют в граф циклы и ситуации, когда в одну вершину входит более одной дуги.

Разумным требованием к процессу применения правил является то, чтобы каждое примененное синтаксическое правило оставалось верным до конца процесса. Это означает, что величины, на которые во время применения правила были наложены определенные ограничения, не должны выходить за рамки этих ограничений при всех последующих применениях правил. В нашей модели элемент пространства категорий

---

[6] Исходные дуги по определению синтаксического правила могут соединять только исходные вершины.

соотносится с узлом мультиструктуры один раз (в рамках исходного словосочетания или в момент применения правила) и больше не меняется. Поэтому единственное, что может измениться, это значения функций $s(v)$ и $e(v)$[7] для узлов мультиструктуры, т.к. среди дуг, добавляемых в процессе применения правил, могут быть дуги, помеченные символом w. Изменение границ узлов, в свою очередь, может нарушить одно из ограничений Ψ.

Для того чтобы исключить такие ситуации было решено дополнить понятие мультиструктуры списком ограничений (многомерных областей), наложенных на значения функций $s(v)$ и $e(v)$ для ее узлов. При применении синтаксического правила дополнительно проверяется, что в результате не нарушаются ограничения из этого списка. После применения правила, та часть его многомерных ограничений, которая относится к функциям $s(v)$ и $e(v)$, переносится на конкретные узлы и добавляется к списку ограничений.

Перейдем к формальному описанию процедуры применения синтаксического правила к мультиструктуре. Пусть задана некоторая мультиструктура $S^m=(W(V=M\cup V',E),g,h,<\Omega_1,V_1>...<\Omega_m,V_n>)$.

Введем следующее обозначение:

$d: V \rightarrow \{(x,y) \mid x,y \in N_0 \cup \{null\}\}$, $d(v) = (s(v),e(v))$

Определение 1.7. Будем говорить, что синтаксическое правило $r(T(V_o\cup V_n, E_o\cup E_n), \alpha_o, \alpha_n, \beta, \Psi)$ *применимо к мультиструктуре* S^m, если можно задать такое биективное отображение $F: V_o \rightarrow V$, что

1. Для любого $v \in V_o$, $g(F(v)) \in \alpha_o(v)$.
2. Для любого $(v_1,v_2) \in E_o$, $r = (F(v_1), F(v_2)) \in E$ и $h(r) = \beta((v_1,v_2))$.
3. Вектор $(g(F(v_1)), g(F(v_2)),..., g(F(v_{|Vo|})), d(F(v_1)), d(F(v_2)),..., d(F(v_{|Vo|}))) \in \Psi$, $v_i \in V_o$, $1 \le i \le |Vo|$
4. Выполнение процедуры применения правила не нарушает пространственных ограничений, наложенных на исходные структуры. Это означает, что для любых i,j вектор $(d(v_1), d(v_2),..., d(v_{|Vij|})) \in \Omega_{ij}$, $v_k \in V_{ij}$
5. После выполнения процедуры применения правила S^m удовлетворяет определению мультиструктуры.

Под *процедурой применения правила* понимается следующая последовательность действий:

1. Добавить к множеству V' новые вершины $v_1', v_2',..., v_{|Vn|}'$
2. Расширить отображение g следующим образом: $g(v_i') = \alpha_n(v_i)(g(F(o_1)), g(F(o_2)),..., g(F(o_{|No|})))$, где $v_i \in V_n$, $o_j \in V_o$
3. Задать вспомогательное отображение F' следующим образом:

---

[7] Следует отметить, что функции $s(v)$ и $e(v)$ вводятся для узлов мультиструктуры совершенно так же, как для узлов синтаксической структуры.

$$F'(v)\begin{cases} F(v), \text{ при } v \in V_o \\ v_i', \text{ при } v = v_i \in V_N \end{cases}$$

4. Добавить к множеству E новые дуги $(F'(v_{1i}), F'(v_{2i}))$, где $(v_{1i}, v_{2i}) \in E_n$, $1 \le i \le |E_n|$

5. Расширить отображение h следующим образом: $h((F'(v_{1i}), F'(v_{2i}))) = \beta(v_{1i}, v_{2i})$, $1 \le i \le |E_n|$

6. Добавить к мультиструктуре пару $<\Omega, (F(v_1), \ldots, F(v_{|V_o|}))>$, $v_i \in V_o$ где Ω – это область Ψ, рассматриваемая на последних $|V_o|$ измерениях.

То, что мультиструктура S_1^m является результатом применения правила r к мультиструктуре S_2^m, мы будем записывать следующим образом: $S_1^m = r(S_2^m)$

Опишем процесс последовательного применения синтаксических правил к мультиструктуре. Используем для этого индуктивное определение.

Определение 1.8. Будем называть *синтаксическим замыканием мультиструктуры* S^m *относительно конечного набора правил* R множество $\overline{S^m}_R$, содержащее все мультиструктуры, которые могут быть получены в процессе применения правил, принадлежащих R, к S^m.

$$\overline{S^m}_R = \left(\bigcup_{k=1}^{\infty} Q_k \right)$$

$$Q_0 = \{S^m\}$$

$$Q_k = Q_{k-1} \cup \{S_i^m \mid \exists r \in R : S_i^m = r(S_j^m)\ S_j^m \in Q_{k-1}\}$$

Определение 1.9. Будем называть *синтаксическим замыканием словосочетания* p *относительно конечного набора правил* R множество $\overline{p_R} = \overline{S^m}_R$, где S^m – исходная мультиструктура словосочетания p.

С учетом введенных понятий *задача синтаксического анализа* формулируется следующим образом: для заданного словосочетания p и заданного набора правил R найти все связные мультиструктуры (синтаксические структуры), принадлежащие $\overline{p_R}$.

Определение 1.10. Будем называть *результатом синтаксического анализа словосочетания* p *относительно конечного набора правил* R множество $R(p) = \left\{S(p) \mid S \in \overline{p_R}\right\}$.

2. Штрафные функции

При работе с реальными системами синтаксических правил, авторы обнаружили, что для многих предложений естественного языка количество элементов множества $R(p)$ оказывается очень большим (может исчисляться тысячами). Причем зависимость количества результатов от длины предложения оказывается экспоненциальной. Это связано, прежде всего, с тем, что с помощью синтаксических правил нет возможности в полной мере учитывать информацию о допустимости тех или иных сочетаний слов (т.к. она существенно зависит от смысла слов). Однако, существует ряд эвристик, позволяющих достаточно эффективно сужать множество результатов, причем эти эвристики не требуют от компьютера «понимания» смысла предложений. Принимая во внимание тот факт, например, что в массе своей предложения естественного языка проективны, можно, в случае наличия в $R(p)$ проективного и непроективного вариантов, второй вариант отбрасывать. Процент случаев, когда такая эвристика окажется верной, очен велик. Для учета подобных явлений в анализатор Treevial был внедрен механизм так называемых *штрафных функций*. Штрафная функция ставит в соответствие любой синтаксической структуре некоторое неотрицательное действительное число, символизирующее степень правдоподобности данной структуры – чем больше число, тем менее вероятно, что данная структура правильная. Конкретные способы задания штрафных функций, используемых анализатором Treevial, описываются в [4].

С учетом штрафной функции можно сформулировать расширенную постановку задачи синтаксического анализа:

Для заданного словосочетания p, заданного набора правил R и заданной штрафной функции P(s) построить набор синтаксических структур s_i *такой, что* $s_i \in \overline{p_R}$, $P(s_i) \le P(s_j)$, *при* $i < j$.

Это означает, что для данного словосочетания требуется не просто найти все возможные синтаксические структуры, но и отсортировать их в порядке возрастания штрафа. Заметим, что важным свойством анализатора, решающего такую задачу, является способность сразу генерировать гипотезы в сортированном или почти сортированном виде. Это позволяет эффективно использовать пороговый штраф – как только появляется гипотеза, штраф которой превышает порог, процесс анализа можно останавливать. На практике такой подход позволяет существенно сократить время анализа и сэкономить вычислительные ресурсы. Анализатор Treevial обладает указанным свойством.

В качестве примера приведем штрафную функцию следующего вида:

$$P(s) = n_{\text{int}}(s) \cdot 20 + n_{frm}(s) \cdot 10,$$

где

$n_{\text{int}}(s)$ – *количество пересечений связей в s*

$n_{frm}(s)$ – *количество обрамлений в s*

Эта функция оценивает то, насколько сильно пересекаются связи синтаксической структуры и то, насколько сильно они друг друга обрамляют.

На рисунке 1 показано, как меняются значения функции при различных аргументах.

Рис.1. Изменение значений штрафной функции в зависимости от синтаксической структуры словосочетания

Заключение

В этой статье была описаны математическая модель синтаксического анализатора Treevial. Модель ориентирована на работу в рамках ключевых подходов к описанию синтаксических структур предложения, таких как деревья зависимостей и системы непосредственных составляющих. Базовая структура, используемая в модели, обобщает обе упомянутые структуры. В работе была сформулирована постановка задачи синтаксического анализа в терминах описанной модели.

Авторами было опробовано несколько подходов к решению поставленной задачи. В основе их всех лежит переборный принцип. Этот факт часто воспринимется как существенный недостаток анализатора. Действительно, в контексте задачи синтаксического

анализа ненаправленный перебор вариантов вряд ли может иметь какое-либо практическое применение. Однако, введение штрафных функций принципиально меняет ситуацию. Процесс перебора становится управляемым.

В данный момент на факультете ВМиК с участием авторов ведутся исследования, которые позволят вычислять штрафные функции не только исходя из простых топологических или статистических критериев, но и с привлечением информации о семантике анализируемых предложений. После окончания этих исследований авторы надеются расширить приведенную модель с помощью новых понятий.

Литература

1. Cunningham a.o. 2002 – H.Cunningham, D.Maynard, K. Bontcheva, V.Tablan. GATE: an Architecture for Development of Robust HLT Applications // Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, July 2002.

2. А.В.Гладкий. Синтаксические структуры естественного языка в автоматизированных системах общения. – М.: Наука, 1985

3. Зализняк, А. А. Словоформа // Лингвистический энциклопедический словарь, М. 1990, с. 470

4. М.Г.Мальковский, А.С.Старостин. Модель синтаксиса в системе морфо-синтаксического анализа «TREETON» // Труды международной конференции Диалог'2006. – М.: изд-во РГГУ, 2006. с. 481-492.

5. Mel'cuk, I.A. Dependency Syntax: Theory and Practice. SUNY Series in Linguistics, Mark Aronoff, series editor. State University of New York Press, Albany, 1988

6. Хомский Н. Синтаксические структуры. – В кн.: «Новое в линвистике», вып. II, 1962, с. 412-526

Старостин А.С.

Синтаксический анализатор Treevial. Алгоритм восходящего синтаксического анализа с памятью, работающий под управлением эвристической функции

Введение

Анализатор Treevial представляет собой программный модуль, написанный на языке Java, предназначенный для автоматического синтаксического анализа текстов на естественном языке. Этот анализатор разрабатывается авторами на кафедре алгоритмических языков факультета ВМиК МГУ, начиная с 2005 года.

Анализатор принимает в качестве входных данных текст на русском языке, снабженный информацией о морфологических показателях, систему синтаксических правил и систему штрафных функций. Последняя используется для того, чтобы осуществлять динамическое ранжирование гипотез. Формальный аппарат описания штрафных функций позволяет оценивать синтаксические структуры без использования информации о конкретных словах, используя лишь «топологические» характеристики структур: уровень проективности, количество зацеплений, наличие обрамлений, гнездований и др[1]. Во многих случаях этих критериев оказывается достаточно для того, чтобы правильная синтаксическая структура оказалась первой в списке возможных вариантов.

Treevial использует стратегию восходящего анализа: в рамках переборного процесса к результатам морфологического анализа применяются синтаксические правила. В результате возникают новые структуры, которые также «поступают на вход» синтаксическим правилам. Таким образом, круг замыкается. Порядок применения правил не фиксируется. Если в процессе удается получить связные древесные структуры, покрывающие все входное предложение, то они считаются результатом синтаксического анализа. Система штрафных функций влияет на перебор – на каждом шаге выбирается определенное правило и производится попытка применить его к наименее штрафованной комбинации подструктур (вычисляется евклидова норма вектора, составленного из значений штрафной функции для структур, образующих комбинацию).

Данная статья является продолжением статьи [*Мальковский М.Г., Старостин А.С. Синтаксический анализатор Treevial. Постановка*

[1] Эти понятия хорошо описаны в [3]

задачи синтаксического анализа], опубликованной в этом сборнике. Там вводятся необходимые понятия и приводится формальная постановка задачи, решаемой анализатором. В первом разделе данной статьи описывается алгоритм работы анализатора. Во втором разделе обсуждаются проблемы, связанные с программной реализацией алгоритма.

1. Алгоритм синтаксического анализа

Первая версия алгоритма, решающего поставленную задачу, описывается в [6]. Приведенный там алгоритм чем-то напоминает переборные алгоритмы игры в шахматы (ср. [1]). В процессе перебора на каждом шаге рассматривается мультиструктура (позиция на доске) с наименьшим штрафом (наиболее перспективная). Применяются допустимые в данный момент правила (делаются ходы) и полученные мультиструктуры (позиции) вновь оцениваются. Следует отметить, что в первой версии алгоритма штрафные функции оценивали мультиструктуры в целом, а не отдельные их компоненты. В функцию включалась не только оценка синтаксических структур (в целом аналогичная используемой сегодня), но и такой параметр, как уровень связности мультиструктуры (количество деревьев, состоящих больше, чем из одного узла).

Существенным недостатком упомянутого алгоритма было то, что в процессе перебора часто применялись одни и те же последовательности правил и, следовательно, тратились лишние вычислительные ресурсы. Одну и ту же составляющую приходилось «собирать» снова и снова, применяя одни и те же правила совершенно одинаковым образом.

Второй проблемой, вытекающей из первой, была проблема так называемых *ранних ошибок*. Так автор называет случаи, когда на каком-то шаге перебора применяется некоторое правило, после чего тратится большое количество времени и ресурсов на перебор комбинаций в контексте примененного правила и все ветви этого переборного процесса приводят к тупику или к сильно штрафованному варианту, причем штраф всегда возникает из-за конфликта с исходным правилом. Другими словами, ошибка допускается рано, а обнаруживается поздно. Такие ошибки часто связаны с проективностью: проводится, например, связь, обрамляющая какой-то небольшой отрезок текста (2-3 слова), после чего система перебирает множество комбинаций во внешней по отношению к отрезку области и каждый раз, проводя связь внутрь этого отрезка, штрафует очередной вариант за непроективность.

Наконец, особой проблемой при использовании алгоритма были длинные предложения, в разных частях которых правила могли

применяться совершенно независимо, что приводило к комбинаторному взрыву. Приходилось искусственным образом ограничивать количество нетривиальных синтаксических структур, входящих в мультиструктуру, что, в свою очередь, могло приводить к тому, что некоторые сложные случаи переставали анализироваться.

Все эти проблемы навели на мысль о существенной модификации алгоритма. Представлялось целесообразным сделать так, чтобы во время перебора те подструктуры, которые уже удалось построить, сохранялись и могли быть дальше использованы как единое целое, без дополнительных вычислений. Похожая идея используется в известном алгоритме CYK ([2],[4],[7]), определяющем, может ли данная строка быть порождена заданной контекстно-свободной грамматикой. Был реализован алгоритм, обладающий указанным свойством. Он используется в последней версии анализатора Treevial.

Основной идеей нового алгоритма является то, что базовой единицей, которая преобразуется, становится уже не мультиструктура целиком, а отдельная синтаксическая структура. Алгоритм работает не со множеством мультиструктур для всего предложения, а со множеством синтаксических структур отдельных «подпредложений». На каждом шаге из множества выбирается и удаляется наименее штрафованная структура, после чего ищутся возможности соединения этой структуры с другими при помощи синтаксических правил. В результате получаются новые синтаксические структуры, которые добавляются к исходному множеству. Если в процессе анализа обнаруживаются структуры, покрывающие все предложение, то они добавляются в конец списка результатов.

Мультиструктуру можно рассматривать как набор синтаксических структур. Для этого достаточно представить словосочетание, на котором построена мультиструктура, в виде объединения словосочетаний, каждое из которых состоит из словоформ, принадлежащих одному из связных фрагментов мультиструктуры.

Синтаксическое правило можно, в свою очередь, рассматривать не как функцию, преобразующую одну мультиструктуру в другую, а как функцию, преобразующую набор синтаксических структур в одну синтаксическую структуру. Единственность результирующей структуры гарантируется тем, что синтаксическое правило является деревом. Следует отметить, что по структуре синтаксического правила можно однозначно определить то, сколько аргументов будет иметь такая функция.

Достаточно просто реализовать алгоритм, который последовательно перебирает все возможные комбинации синтаксических структур, выделяет из них те, что покрывают все входное предложение, и, наконец, сортирует их в соответствии со штрафной функцией *P(s)*. Однако, необходимость ждать окончания

работы алгоритма, чтобы воспользоваться результатами анализа, является слишком серьезным ограничением. Дело в том, что при работе с большими предложениями и сложными системами правил, полный анализ может занимать существенное время, причем большая его часть будет тратиться на перебор гипотез с очень высокими значениями штрафной функции.

Поэтому была предложена новая схема перебора, гарантирующая, что выходной поток результатов (синтаксических структур, покрывающих входное предложение) будет отсортирован относительно $P(s)$. Это означает, что про любой результат, сгенерированный схемой, можно будет утверждать что значение штрафной функции для него выше чем значение штрафа для всех результатов, сгенерированных ранее. Выполнение такого условия позволяет использовать результаты анализа, не дожидаясь завершения работы алгоритма. Кроме того, появляется возможность отсекать гипотезы в соответствии с некоторым пороговым значением штрафа. Это часто бывает нужно при решении практических задач с использованием анализатора.

Для того, чтобы указанное свойство переборного алгоритма могло быть выполнено, необходимо наложить на функцию $P(s)$ и систему синтаксических правил следующее ограничение: для любого правила $r \in R$ должно быть верно, что для любых s_i из того, что $s_r = r(s_0, s_1, s_2, \ldots, s_n)$, следует, что $P(s_r) \geq max\ P(s_i)$ $(i=0..n)$. Другими словами, все синтаксические правила должны быть функциями, неубывающими относительно функции $P(s)$. На практике выполнения этого условия можно достичь, пожертвовав некоторыми сложными с лингвистической точки зрения случаями, редко встречающимися в реальных текстах. Следует отметить, что проводились эксперименты и с системами правил и штрафных функций, для которых описанное условие не выполняется в общем случае, но выполняется в большинстве реально встречающихся случаев. Список результатов синтаксического анализа при работе с такими системами оказывается «почти сортированным», чего все равно оказывается вполне достаточно для многих практических задач.

Приведем описание алгоритма анализа, опирающегося на указанное ограничение. Пусть задано исходное словосочетание $p=(m_1,\ldots,m_n f)$, система синтаксических правил R и штрафная функция $P(s)$.

Введем следующие обозначения:

S – множество синтаксических структур.

Перед началом алгоритма $S = \{s=(T(V=\{m_i\},\varnothing),f,\varnothing) \mid i=1..n\}$

D – множество обработанных комбинаций вида $(s_0, s_1, s_2, \ldots, s_n, r)$, т.е. комбинаций некоторого набора синтаксических структур и

одного синтаксического правила. На начальном шаге множество D пусто.

B – вспомогательное множество синтаксических структур.

$R(p)$ – список результатов (изначально пустой).

$N_P(s_0, s_1, s_2, ..., s_n) = |(P(s_0), P(s_1), P(s_2), ..., P(s_n))|$ – евклидова норма вектора, составленного из значений штрафной функции для элементов некоторой комбинации синтаксических структур.

Схема работы алгоритма такова:

1. Из всех комбинаций $c=(s_0, s_1, s_2, ..., s_m, r)$, $s_i \in S$, $s_i \neq s_j$ при $i \neq j$, $r \in R$ таких, что $c \notin D$ выбрать комбинацию $c'=(s'_0, s'_1, s'_2, ..., s'_m, r')$, для которой $N_P(s'_0, ..., s'_n)$ минимально. Если ни одной комбинации не нашлось, завершить алгоритм.
2. Выбрать из множества B элемент s_b, для которого значение штрафной функции минимально. Если такого элемента нет, перейти к шагу 1. Если $P(s_b) < N_P(s'_0, ..., s'_n)$, то удалить s_b из B, добавить в $R(p)$ и перейти к шагу 2. В противном случае перейти к шагу 3.
3. Проверить допустимость структуры $s_r=r(s'_0, s'_1, s'_2, ..., s'_m, r')$. Если структура допустима, то добавить ее в множество S. Если нет, перейти к шагу 1.
4. Если s_r покрывает все исходное предложение, то добавить s_r в B.
5. Добавить c' в множество D и перейти к шагу 1.

После завершения работы алгоритма список $R(p)$ будет содержать решение задачи синтаксического анализа.

2. Программная реализация

Самым сложным с вычислительной точки зрения шагом описанного в предыдущем разделе алгоритма является шаг №1. Для того чтобы выбор комбинации с оптимальной нормой выполнялся эффективно, были реализованы специальные структуры данных – комбинаторы. Комбинатор представляет собой объект с n входами и одним выходом. Входы комбинатора являются сортированными относительно штрафной функции множествами синтаксических структур. На выходе комбинатор генерирует различные комбинации из n синтаксических структур. При этом выполняются следующие условия:

1. Элемент комбинации с номером i принадлежит входному потоку с номером i.
2. Комбинации никогда не повторяются.
3. В каждый момент времени на выходе комбинатора находится комбинация с наименьшим из возможных на данный момент значением N_P.

Важным свойством комбинатора является то, что он способен динамически реагировать на изменения во входных потоках. Несколько упрощая можно сказать, что если в каком-то входном потоке появляется «хороший» элемент, то комбинатор автоматически переключается и начинает генерировать комбинации с этим элементом.

Перед началом анализа система синтаксических правил преобразуется в систему комбинаторов. Это преобразование может осуществляться различными способами. Самый простой принцип следующий: с каждым правилом соотносится набор входных сортированных потоков в соответствии с количеством аргументов правила и один комбинатор. Могут быть и более сложные варианты. Например, если наложить на весь переборный процесс запрет на создание разрывных структур и ограничиться правилами размерностью не больше 3, то становится возможным организовать систему комбинаторов, в которой каждый элемент будет комбинировать структуры, сконцентрированные вокруг определенного места в тексте. Возможны и другие принципы построения системы комбинаторов.

Для того, чтобы на каждом шаге была выбрана комбинация с минимальной нормой, требуется из набора текущих выходов комбинаторов выбирать лучший. Для этого уже сами комбинаторы организуются в список, который динамически поддерживается в сортированном состоянии. Заметим, что из того, что отдельные комбинаторы гарантируют неповторяемость комбинаций, следует неповторяемость комбинаций во всей системе в целом (ср. множество D в описании алгоритма).

Для того, чтобы возникающие синтаксические структуры эффективно распределялись по входным потокам комбинаторов, был реализован специальный модуль, динамически индексирующий структуры.

Анализатор Treevial написан на языке Java. Для эффективного выполнения второго шага описанного алгоритма, множество B было реализовано как сортированная очередь (был использован класс PriorityBlockingQueue из пакета java.util.concurrent, ср. [5]). Следует отметить, что на практике объем этой очереди оказывается незначителен, т.к. она постоянно опустошается.

На рисунке 1 изображена схема работы синтаксического анализатора.

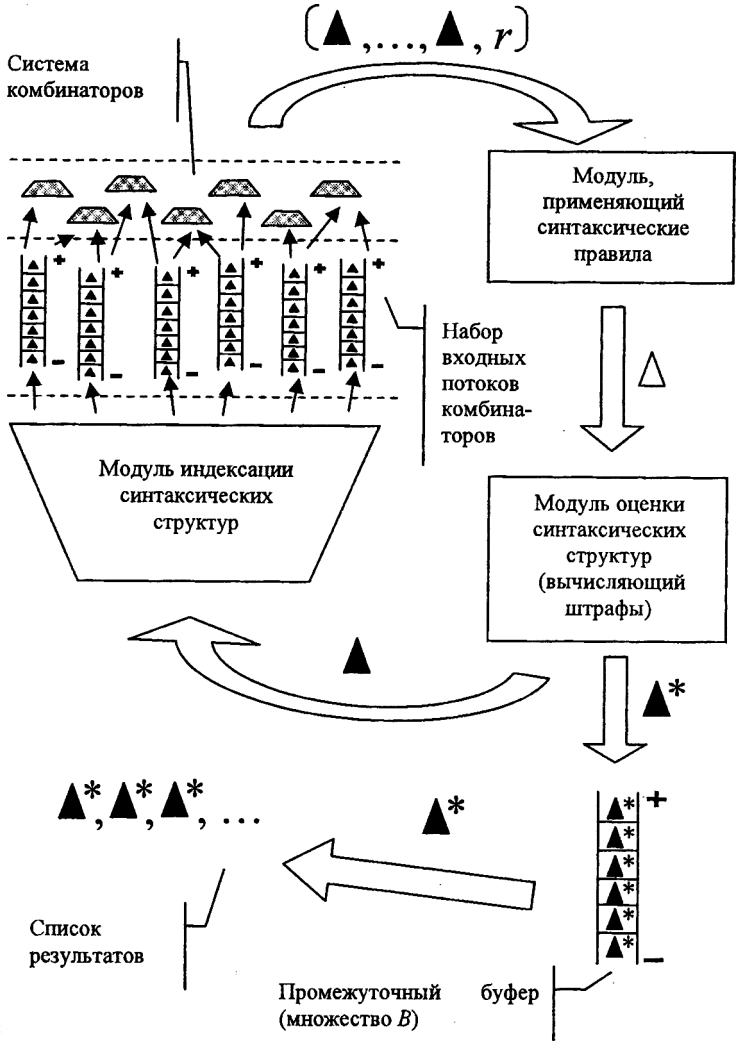


Рис. 1. Схема работы синтаксического анализатора. Треугольники обозначают синтаксические структуры. Прозрачные треугольники обозначают структуры, для которых еще не посчитан штраф. Треугольники со звездочкой обозначают структуры, покрывающие все входное предложение

В целях экономии синтаксические структуры не хранятся в памяти в виде деревьев. Каждая синтаксическая структура хранится как пара *<примененное правило, массив синтаксических структур>*. В тот момент, когда требуется произвести какое-то алгоритмически нагруженное действие над синтаксической структурой (например, посчитать ее штраф), интерпретация копируется в специальную структуру, построенную на хэш-таблицах и оптимизированную для эффективной работы с деревьями.

Заметим, что неповторяемость комбинаций не гарантирует неповторяемость синтаксических структур, т.к. одну и ту же синтаксическую структуру часто можно «собрать» более чем одним способом, применяя правила в разном порядке. Однако, в описании алгоритма фигурирует именно множество синтаксических структур S, следовательно, элементы не должны повторяться. Для того чтобы не допускать повторений структур в ходе перебора, используются хэш-таблицы. Синтаксические структуры удается эффективно хэшировать, т.к. каждая структура однозначно определяется набором примененных правил. Хэш-функция строится по этому набору.

Заключение

В статье был описан алгоритм синтаксического анализа, используемый анализатором Treevial. Отличительной чертой алгоритма является то, что в процессе анализа используется эвристическая штрафная функция, управляющая переборным процессом. Второй важной чертой алгоритма является то, что сформированные гипотезы о синтаксическом строении частей предложения могут повторно использоваться в процессе перебора.

Следует отметить, что развитие синтаксического анализатора возможно в двух направлениях. Во-первых, можно распространить основные принципы анализа на более низкие языковые уровни. К примеру, можно себе представить систему морфологических правил, на вход которой будет поступать гипотетическое морфонологическое покрытие (набор возможных морфем). Правила комбинировали бы морфемы и в результате получались бы варианты морфологического анализа. Другими словами, представляется, что архитектура анализатора в пределе могла бы быть обобщена на все, что, используя терминологию де Соссюра, принято называть означающим. Второе направление развития анализатора связано, наоборот, с означаемым. Кажется разумным организовать активное взаимодействие синтаксического анализатора с семантическим модулем. Этот модуль должен уметь динамически строить семантические представления для любой синтаксической структуры, словоформы или морфемы. Если он,

кроме того, будет способен оценивать эти структуры (аналогично системе штрафных функций на синтаксическом уровне), то эти оценки можно будет динамически учитывать на более низких уровнях, что должно существенно повысить качество анализа. В данный момент на факультете ВМиК с участием автора ведутся исследования, целью которых является создание прототипа такого модуля.

Литература

1. Ботвинник М. М. Алгоритм игры в шахматы. – М.: Наука, 1968

2. John Cocke And Jacob T Schwartz (1970). Programming languages and their compilers: Preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University.

3. А.В.Гладкий. Синтаксические структуры естественного языка в автоматизированных системах общения. – М.: Наука, 1985

4. T. Kasami (1965). An efficient recognition and syntax-analysis algorithm for context-free languages. Scientific report Afcrl-65-758, Air Force Cambridge Research Lab, Bedford, MA.

5. Doug Lea, Concurrent Programming in Java, Addison-Wesley, Reading, Massachusetts, October 1996

6. М.Г.Мальковский, А.С.Старостин. Модель синтаксиса в системе морфо-синтаксического анализа «TREETON» // Труды международной конференции Диалог'2006. – М.: изд-во РГГУ, 2006. с. 481-492.

7. Daniel H Younger (1967). Recognition and parsing of context-free languages in time n3. Information and Control 10(2): 189-208.

Раздел IV.

Сообщения

Балаханов В.А., Кокарев В.А.

Муравьиный алгоритм построения статико-динамических расписаний и исследование его эффективности

Введение

Статико-динамическое расписание представляет собой набор окон, в рамках которых выполняются все работы. При этом распределение работ по окнам задается статически, в то время как время выполнения той или иной работы в рамках одного окна определяется динамически в процессе работы системы. Использование таких расписаний при работе систем реального времени позволяет уменьшить время реакции на прерывания и переключение режимов работы. Однако системы реального времени накладывают дополнительные ограничения на расписание. Кроме времени выполнения, каждая работа характеризуется также директивным интервалом, в пределах которого допустимо ее выполнение. Эти особенности, а также различные дополнительные ограничения, накладываемые на расписание используемыми системами, делают практически неприменимыми для решения реальных задач построения статико-динамических расписаний, как классические итерационные алгоритмы, так и различные конструктивные эвристики.

Алгоритмы оптимизации, имитирующие кооперативное поведение муравьев в колонии, впервые были предложены в 1992 году для решения задачи коммивояжера [1]. Впоследствии муравьиные алгоритмы были успешно использованы для решения различных комбинаторных задач, в том числе ряда задач построения расписания [2,3,4]. Муравьиные алгоритмы сочетают в себе конструктивный способ построения решения с итерационным исследованием пространства решений. Это делает муравьиные алгоритмы более гибкими при решении сложных комбинаторных задач. Однако при этом на работу муравьиных алгоритмов влияет большое число параметров, требующих тонкой настройки. В данной работе будет описан муравьиный алгоритм

решения задачи построения статико-динамических расписаний на примере систем, работающих по стандарту ARINC-653 [5], и приведены результаты исследования его эффективности.

1. Задача построения статико-динамического однопроцессорного расписания

Статико-динамическое расписание представляет собой набор окон, каждое из которых характеризуется временем открытия, временем закрытия и списком работ, выполняющихся внутри окна. При этом порядок выполнения работ внутри окна определяется динамически и заранее неизвестен. Длина окна должна быть не меньше, чем суммарное время выполнения работ внутри него. Наряду с заданным списком прикладных задач в вычислительной системе могут выполняться системные задачи, список которых заранее неизвестен; на выполнение этих задач в каждом окне должен быть отведен определенный резерв времени. При этом временной интервал окна должен лежать внутри каждого из директивных интервалов работ, выполняющихся в данном окне, чтобы гарантировать, что директивные интервалы работ не будут нарушены.

Примером систем реального времени, использующих статико-динамические расписания, являются системы, работающие по стандарту ARINC-653. Одним из основных понятий стандарта является понятие раздела: в системе имеется определенный набор разделов, и каждая работа характеризуется номером раздела. Работы из одного раздела могут выполняться непосредственно одна за другой, без задержек; для перехода из одного раздела в другой необходимо переключение контекста, которое требует определенного времени. Все работы внутри окна должны принадлежать одному разделу, таким образом, работы внутри окна могут выполняться непосредственно друг за другом, без временных затрат на переключение контекста, которое может производиться только между закрытием одного окна и открытием следующего. Таким образом, для каждой работы задано время выполнения, номер раздела и директивный временной интервал выполнения.

В данной работе будут рассматриваться системы, работающие в соответствии со стандартом ARINC-653. Приведем формальную постановку задачи построения расписания для таких систем:

Задано множество работ:

$SW = \{a_i =< s_i, f_i, t_i, p_i > | i \in [1..n]\}$, где

- $[s_i, f_i)$ – директивный временной интервал;

- t_i – время выполнения работы;
- p_i – номер раздела работы

В дальнейшем будем предполагать, что $s_i < f_i$ и $t_i \le f_i - s_i$.

Кроме того, заданы два параметра: $\Delta 1$ и $\Delta 2$, определяющие, соответственно, временные затраты на переключение контекста между окнами и количество времени, отводимое внутри каждого окна на системные задачи.

Требуется построить расписание, представляющее собой набор окон и распределение работ по окнам:

$$SP = < W, D >; W = \{w_i = < S_i, F_i > | i \in [1..m]\};$$

$D = \{d_i | i \in [1..n], d_i \in [0..m]\}$, где

- $[S_i , F_i)$ – временной промежуток между открытием и закрытием окна;
- d_i – номер окна, в котором выполняется i-я работа (если d_i=0, то работа считается неразмещенной).

Будем предполагать, что $S_i < F_i$, а окна упорядочены в порядке возрастания времени открытия S_i.

1. $\forall i \in [1..n], \forall j \in [1..m]: d_i = j \Rightarrow s_i \le S_j \le F_j \le f_i$ – временной интервал окна лежит внутри директивных интервалов работ, выполняющихся в окне;
2. $\forall i, j \in [1..n]: d_i = d_j \Rightarrow p_i = p_j$ – разделы работ, размещенных внутри одного окна, совпадают;
3. $\forall i, j \in [1..m]: i < j \Rightarrow S_j \le F_i + \Delta 1$ – окна не пересекаются и между любыми двумя окнами есть промежуток не короче времени, необходимого на переключение контекста;
4. $\forall j \in [1..m]: \sum_{i: d_i = j} t_i + \Delta 2 \le F_j - S_j$ – суммарное время выполнения всех работ из одного окна и выполнения системных задач не больше, чем длина окна.

В качестве критерия оптимальности рассмотрим отношение количества размещенных работ к общему количеству работ: $T(SP) = \sum_{i=1}^{n} \delta_i$, где $\delta_i = \begin{cases} 1, если\ d_i \neq 0 \\ 0, если\ d_i = 0 \end{cases}$.

Рассматриваемая задача принадлежит к классу NP-трудных задач. Это можно доказать сведением к ней NP-полной задачи построения однопроцессорного расписания для работ с произвольными директивными интервалами [6].

2. Описание алгоритма

Основными задачами, которые необходимо решить для того, чтобы использовать муравьиный алгоритм для решения конкретной задачи оптимизации, являются:

1. Сведение задачи оптимизации к задаче нахождения на графе маршрута, обладающего определенными свойствами.
2. Задание глобальной целевой функции на множестве маршрутов.
3. Задание локальной эвристической функции на ребрах графа.
4. Определение алгоритма построения маршрута муравьем (например, определение правила формирования табу-списка вершин).

Построим полносвязный граф, каждой вершине которого будет соответствовать одна из размещаемых работ: $G=<N=\{n_i \mid i\in[1..n]\}\cup\{O\}, A=\{(n_i,n_j)\mid i,j\in[1..n], i\neq j\}\cup\{(O,n_i)\mid i\in[1..n]\}>$. Кроме того, добавляется еще одна вершина O, соответствующая началу расписания. Потребуем, чтобы муравьиный алгоритм строил маршруты, начинающиеся в вершине O и проходящий через каждую вершину ровно один раз. Каждому такому маршруту соответствует последовательность работ, в которую включены все работы из исходного набора, причем каждая работа включена в данную последовательность ровно один раз. В качестве целевой функции рассмотрим отношение количества работ, размещенных в расписание без нарушения условий корректности, к количеству исходных заданных работ. Значение данной функции вычисляется алгоритмом, строящим расписание по маршруту [7].

Муравьиные алгоритмы являются итерационными, на каждой итерации алгоритма производится одновременный поиск маршрута несколькими муравьями. Общая схема работы муравьиных алгоритмов представима в следующем виде:

1. Задание начальных значений параметров алгоритма и количества феромона на ребрах графа.
2. Определение количества и начального положения муравьев.
3. Поиск муравьями решений в соответствии с заданным алгоритмом построения маршрута.
4. Обновление количества феромона на ребрах в зависимости от качества полученных решений.
5. Если условие останова не выполнено, то переход к п.2.

В предложенном алгоритме количество муравьев равно количеству размещаемых работ, все они изначально помещаются в вершину O. Муравьи строят маршрут, последовательно переходя из одной вершины к следующей. На выбор очередной вершины влияют состояние табу-списка

(в который добавляются все пройденные данным муравьем вершины), количество феромона на ребре, ведущем из текущей вершины в следующую, и значение локальной эвристической функции на данном ребре. Вероятность перехода *k*-го муравья из *i*-й вершины в *j*-ю на *t*-й итерации алгоритма рассчитывается по следующей формуле:

$$P_{ij,k}(t)=\begin{cases}\dfrac{(\tau_{ij}(t))^{\alpha}\cdot(\eta_{ij}(t))^{\beta}}{\sum_{l\in J_k}(\tau_{il}(t))^{\alpha}\cdot(\eta_{il}(t))^{\beta}}, j\notin J_k\\ 0, j\in J_k\end{cases} \quad (1)$$

В данной формуле $\tau_{ij}(t)$ - количество феромона на ребре (i,j), $\eta_{ij}(t)$ – значение локальной эвристической функции на этом ребре, α и β - параметры алгоритма, а J_k – табу-список вершин для муравья k. Выбор очередной вершины осуществляется по «правилу рулетки» с использованием вероятностей, вычисленных по формуле (1). Построение маршрута завершается, когда не осталось вершин, не включенных в маршрут.

После того, как все муравьи завершили построение маршрутов, на ребра, входящие в построенные маршруты, добавляется количество феромонов, зависящее от качества полученного решения:

$$\Delta\tau_{ij,k}(t)=\begin{cases}F(T_k(t)),(i,j)\in T_k(t)\\ 0,(i,j)\notin T_k(t)\end{cases} \quad (2)$$

Здесь $T_k(t)$ – маршрут, построенный *k*-м муравьем, а $F(T)$ – значение целевой функции. Таким образом, правило, по которому обновляется количество феромона на $(t+1)$-й итерации на ребре (i,j), принимает следующий вид:

$$\tau_{ij}(t+1)=(1-p)\cdot\tau_{ij}(t)+\sum_{k=1}^{m}\Delta\tau_{ij,k}(t) \quad (3)$$

В этой формуле m – количество муравьев в колонии, а p – коэффициент испарения феромонов.

В качестве критерия останова будем рассматривать ограничение на количество итераций без улучшения решения.

В качестве локальной целевой функции на ребрах графа G используется следующая функция: $\eta_{ij}=\begin{cases}1/(\theta+1), если\ \theta\geq 0\\ 0, если\ \theta<0\end{cases}$, где $\theta=(f_j-t_j)-(s_i+t_i)$ – разница во времени между минимальным временем завершения *i*-й работы и максимальным временем начала *j*-й работы. Если $\theta < 0$, то *j*-я работа не может идти в расписании после *i*-й. В противном случае, чем меньше значение θ, тем меньше максимальный возможный промежуток в расписании между работами, соответствующими вершинам i и j, и тем больше значение локальной целевой функции.

3. Исследование алгоритма

На эффективность работы муравьиных алгоритмов влияет большое количество параметров. В данном разделе приведены результаты исследования зависимости качества полученных решений от параметров алгоритма, а также от характеристик исходных данных.

Для проведения численных исследований использовалась следующая методика:

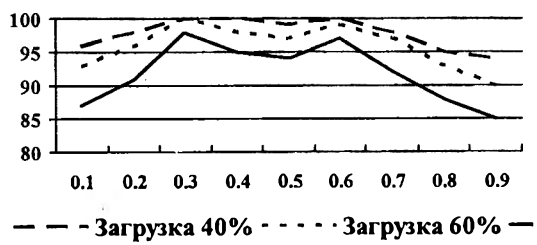
1. Генерация расписания с заданными характеристиками.
2. Построение набора исходных данных по созданному расписанию.
3. Запуск алгоритма на построенных исходных данных.

Данный подход позволяет оценить эффективность алгоритма на исходных данных с различными характеристиками, при этом гарантируется существование корректного расписания, в которое включены все размещаемые работы. В качестве характеристик исходных данных рассматривались загрузка процессора (которая вычисляется как отношение суммарной длительности работ к длине интервала планирования), количество работ и разделов. Для каждого набора характеристик производилось 100 запусков алгоритма на различных исходных данных, полученные результаты усреднялись.

Исследования показали, что параметры алгоритма влияют на качество решения независимо от характеристик исходных данных и друг от друга. Это означает возможность настройки этих параметров независимо друг от друга. В данной работе приводятся результаты исследований качества полученных решений в зависимости от значений следующих параметров, оказывающих наибольшее влияние на качество решений.

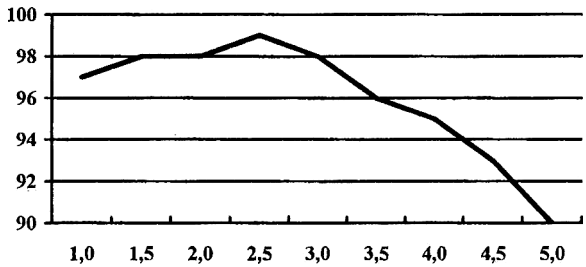
На диаграмме 1 приведены графики зависимости качества решений (определяемого как процент размещенных работ) от значения коэффициента испарения p при различной загрузке процессора. Здесь и далее количество размещаемых работ – 60, количество разделов – 5. Наилучшие результаты были получены при значении p=0.3.

Диаграмма 1. Зависимость качества решений от коэффициента *p*



Исследования показали, что разработанный алгоритм очень часто строит расписание, в которое не включено малое число работ (около 1-5%), хотя при этом расписание, в которое включены все работы, существует. Было выдвинуто предположение, что данный недостаток алгоритма проявляется из-за используемой целевой функции. Данная целевая функция линейно зависит от числа размещенных в расписание работ, и ее значение незначительно различается для расписаний, в которые включены почти все работы. Как следствие, алгоритм не стремится улучшать расписания со значением целевой функции, близким к оптимальному. Для решения этой проблемы предлагается использовать целевую функцию с большим различием в значениях для расписаний, в которые включены почти все работы. Предложенному условию удовлетворяют функции вида $T_\lambda(SP) = (T(SP))^\lambda$ при значениях $\lambda > 1$. На диаграмме 2 приведен график зависимости качества решений от значения параметра λ при загрузке процессора 60%. Наилучшие результаты были получены при значении $\lambda = 2.5$.

Диаграмма 2. Зависимость качества решений от параметра λ



На построение решения муравьиным алгоритмом влияют два механизма: использование локальной эвристической функции и феромон, который представляет собой память алгоритма, в которой хранится информация о том, какие участки входят в наилучшие найденные маршруты. Их взаимное влияние на построение маршрута определяется в формуле (1) параметрами α и β, соответственно для феромона и локальной эвристики. При $\alpha = 0$ муравьи стремятся выбрать вершину, к которой ведет ребро с максимальным значением локальной эвристической функции. При $\beta = 0$ работает только поиск при помощи феромонов.

В таблице 1 приведены результаты численных исследований влияния феромона и локальной эвристической функции на качество

решения. Показана зависимость качества решений, полученных различными модификациями алгоритма, от загрузки процессора.

Таблица 1. Исследование влияния феромона и локальной эвристики

Тип алгоритма	Загрузка процессора			
	30%	50%	70%	90%
Оптимальные значения $(\alpha = 3, \beta = 1)$	100	100	98	95
Поиск при помощи локальной эвристики $(\alpha = 0, \beta = 1)$	53	50	50	50
Поиск при помощи феромонного следа $(\alpha = 1, \beta = 0)$	33	37	38	38
Случайный поиск $(\alpha = \beta = 0)$	34	37	38	39

Как видно из таблицы, поиск при помощи только феромонного следа дает те же результаты, что и случайный поиск. Это связано с тем, что на первых итерациях алгоритма поиск ведется случайным образом, не опираясь на локальную эвристику, и в дальнейшем алгоритм стабилизируется на первых найденных решениях. Однако использование сочетания поиска при помощи феромонного следа и локальной эвристики дает существенное улучшение решений по сравнению с поиском только при помощи локальной эвристики.

4. Заключение

Исследования эффективности разработанного алгоритма показали, что муравьиные алгоритмы в целом применимы для решения поставленной задачи. Однако показано, что разработанный алгоритм способен разместить все работы только при небольших значениях загрузки (до 50%). Кроме этого, результаты, показываемые алгоритмом, нестабильны: при одних и тех же характеристиках исходных данных разброс значений целевой функции для получаемых решений составляет 3-5%. Перспективным направлением развития является разработка алгоритмов построения расписания по найденному маршруту на основе ограниченного перебора и анализа наиболее загруженных участков расписания. Кроме этого, важным вопросом является разработка модификаций алгоритма, направленных на расширение области поиска алгоритма при одном запуске, что позволит получать более стабильные результаты.

Список литературы

1. Dorigo M. Optimization, Learning and Natural Algorithms. // PhD Thesis. Dipartimento di Elettronica, Politechnico Di Milano, Milano. 1992.

2. Ritchie G. Static Multi-processor Scheduling with Ant Colony Optimization and Local Search. // Master's Thesis. University of Edinburgh, Edinburgh. 2003.

3. Blum C., Sampels M. Ant Colony Optimization for FOP Shop Scheduling: A case study on different pheromone representation // In Proceedings of the 2002 Congress on Evolutionary Computations, Honolulu. 2002.

4. Гафаров Е.Р. Гибридный алгоритм решения задачи минимизации суммарного запаздывания для одного прибора // Москва, ВЦ РАН. 2006.

5. Arinc Specification 653. Airlines Electronic Engineering Committee. [PDF] (http://www.arinc.com).

6. Гэри М., Джонсон Д. Вычислительные машины и трудно решаемые задачи. – М.: Мир, 1982. 416 с.

7. Балаханов В.А., Костенко В.А. Способы сведения задачи построения статико-динамического однопроцессорного расписания для систем реального времени к задаче нахождения на графе маршрута // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007. – С.148-156

Брусенцов Н.П.

О содержательном истолковании логико-алгебраических выражений

Содержательная алгебра логики предполагает не формальную логику высказываний и предикатов, а логику реальных вещей, определяемых совокупностями присущих и антиприсущих им *качеств*. Термины x, y, z, ... обозначают первичные (несоставные) качества. Присущности и антиприсущности терминов суть *существенные особенности* вещей. *Сущность* вещи представляется конъюнкцией ее особенностей, в которой антиприсущие термины снабжены штрихами, а *несущественные* (*привходящие*) особенности умалчиваются.

Например, конъюнкция $xy'w$ на наборе терминов x, y, z, w выражает сущность $xy'w$-вещи, которой присущи качества x и w, антиприсуще y, а качество z несущественно, может как быть, так и не быть.

Содержательно "конъюнкция" выражает *совместность*, а "дизъюнкция" - *сосуществование*, как показывает приведенный пример:

$$xy'zw \vee xy'z'w = xy'(z \vee z')w = xy'w$$

Дизъюнкция сущностей выражает особенность, присущую каждой из вошедших в нее вещей и обозначающую *класс*, включающий все эти вещи. Булева алгебра, удовлетворяя закону исключенного третьего, не допускает в своих классах привходящих вещей - не включенные в дизъюнкцию вещи из класса необходимо исключены. В такой алгебре при отображении даже простейших реальных отношений возникают парадоксы.

Чтобы восстановить адекватность алгебры и отображаемой ею логики, необходимо упразднить из нее закон исключенного третьего, распространив существующий для первичных особенностей статус привходящего на составные особенности - конъюнкции, определяющие вещи и классы вещей.

Подобно несущественным терминам в конъюнкциях, привходящие вещи в дизъюнкциях, определяющих классы, надлежит умалчивать, а исключаемые - отмечать аналогичным штриху символом - знаком минус. Например, отношение необходимого следования, вырожденное в двухзначной алгебре в парадоксальную материальную импликацию:

$$x \to y = xy \vee x'y \vee x'y' = x' \vee y\ ,$$

в усовершенствованной предложенным образом трехзначной алгебре получается содержательно безукоризненным [1]:

$$x \Rightarrow y = xy \vee -xy' \vee x'y' = x \vee -xy' \vee y'.$$

Поскольку из данного класса xy'-вещи исключены, то каждая x-вещь необходимо будет xy-вещью, а каждая y'-вещь не может не быть $x'y'$-вещью, т.е. из x необходимо следует y, а из y' необходимо следует x'. Невхождение в дизъюнкцию $x'y$-вещи означает ее несущественность для представленного отношения: из x' не следует с необходимостью y, как и из y не следует x'.

Таково содержательное истолкование трехзначного выражения алгебры классов, представляющее контрапозитивное отношение непарадоксального необходимого следования $x \Rightarrow y$, оно же $y' \Rightarrow x'$. Графически это отношение отображается трехзначной таблицей истинности Пирса. Оказывается, Пирс в 1909 г. обобщил свою общепринятую двухзначную таблицу, приведя ее в соответствие с непарадоксальной силлогистикой Аристотеля. Но логики-двухзначники не восприняли этой фундаментальной коррекции их неадекватной науки, непреклонно соблюдая нелепый закон исключенного третьего.

Трехзначным аналогом пирсовой таблицы истинности является предложенная в 1870-х годах диаграмма Льюиса Кэррола [2], формально идентичная той же таблице, но истолковываемая не как класс, а как множество вещей (не экстенсионально, а интенсионально). Алгебраически отношение необходимого следования в кэрроловом истолковании выражается конъюнкцией существований, несуществований и несущественности вещей:

$$x \Rightarrow y \equiv \mathrm{V}xy\mathrm{V}'xy'\ \mathrm{V}x'y' \equiv \mathrm{V}x\mathrm{V}'xy'\mathrm{V}y'.$$

Данному множеству принадлежат (существуют в нем) xy- и $x'y'$-вещи, антипринадлежат xy'-вещи, а неупомянутые в выражении $x'y$-вещи к нему не относятся, несущественны. Такое трехзначное описание избавляет теорию множеств от свойственных ей парадоксов.

К сожалению, диалектическое мышление Кэррола, как и основоположника адекватной логики Аристотеля, все еще непостижимо для формалистов-двухзначников, даже изобретающих бессодержательные трехзначные импликации. А по сути ведь следование y из x необходимо, если сущность y содержится в сущности x.

Литература

1. Брусенцов Н.П., Владимирова Ю.С. Конструктная компьютеризация силлогистики.// Математические методы распознавания образов (ММРО-13) -М.: МАКС-Пресс, 2007. С.10-13.

2. Льюис Кэррол. Символическая логика. // Кэррол Л. История с узелками / Пер. с англ. -М.: «Мир», 1973. С.189-361.

Владимирова Ю.С.

Силлогистика Аристотеля и гераклитов принцип сосуществования противоположностей

Принятый в современной математической логике формальный подход не предполагает раскрытия сущностей рассматриваемых вещей. Напротив, материальная импликация, моделирующая в логике отношение следования, допускает логическое отношение между содержательно никак не связанными высказываниями, которые к тому же могут быть противоречивыми, что объясняется необходимостью математических применений [1, с.79]. В то же время практическая полезность логического исчисления обуславливается адекватностью представления в нем содержательного следования, являющегося основанием любого рассуждения, поэтому недостаточность представления следования в двузначной логике обнаруживается отсутствием удовлетворительных решений задачи автоматизации рассуждения.

Аристотелева силлогистика, неоправданно считающаяся «узкой системой» [2, с. 189] и даже не имеющей «существенного значения с точки зрения истории математической логики» [3, с. 57], не поддается адекватной формализации средствами двухзначной логики в первую очередь за счет отсутствия в последней возможности выражения привходящего, не необходимого статуса вещи. Но даже и попытки выразить силлогистику средствами трехзначных и модальных исчислений не дали должного результата.

Причиной этого в первую очередь является неформальный характер аристотелевой силлогистики: предмет ее – не абстрактные высказывания, а сущности вещей, охарактеризованных посредством двухзначных критериев, на взаимосвязи которых основывается логический вывод. Отношение содержательного необходимого следования представлено в силлогистике общеутвердительной посылкой «Все x суть y», определенной Аристотелем в полном соответствии здравому смыслу. Возможность заключать что-либо из несуществующего или выводить из чего-либо общезначимое исключена принятием принципа сосуществования противоположностей [4], естественного с точки зрения достоверного рассуждения, и заключающегося в необходимости совместного существования вещей, которым присущи рассматриваемые термины, и вещей, которым эти термины антиприсущи: x-вещи должны сосуществовать с x'-вещами, y-вещи - с y'-вещами и т.д. При соблюдении этого принципа отношение содержательного следования y из x необходимо соблюдено, если не существуют xy'-вещи. Существование хотя бы одной xy'-вещи делает

следование y из x невозможным, когда xy'-вещи не являются необходимыми, то и следование y из x возможно, но не необходимо.

Вещь в силлогистике характеризуется присущностью ей обозначенных терминами x, y, z, ... качеств или их антиподов x', y', z', ..., т.е. ее особенностями. Существование вещей, обладающих особенностью x (x-вещей) обозначается префиксным дизъюнктом Vx, а существование обладающих антиподом особенности x x'-вещей – дизъюнктом Vx'. Несуществование, например, x-вещей обозначается инверсией соответствующего дизъюнкта $V'x$. Силлогистика оперирует совокупностями различных совместно существующих вещей. Например, в совокупности $VxyV'xy'V'x'yVx'y'$ совместно существуют xy и $x'y'$-вещи, а xy'- и $x'y$-вещи отсутствуют.

Достоверное представление силлогистических посылок требует наличия наряду со статусами существования и несуществования вещей третьего-привходящего статуса – «не исключено», «возможно». Общеутвердительная силлогистическая посылка представляется конъюнкцией трех дизъюнктов:

$$Axy \equiv VxyVxy'Vx'y'.$$

Действительно, y необходимо следует из x, если не существует xy'-вещей, xy- и $x'y'$-вещи существуют с необходимостью, а $x'y$-вещи могут существовать, но не необходимы [5, с. 215, 57в1]. Умалчивание в приведенном отношении дизъюнкта $Vx'y$ придает выражению диалектический характер. Добавление этого дизъюнкта означало бы необходимое существование $x'y$-вещей, что изъяло бы из рассуждения оперирование неконстантными совокупностями, являющихся по сути предметом всякого рассуждения. Требование несуществования $x'y$-вещей превращает приведенное выражение в выражение отношения эквивалентности x и y. Неисключенность $x'y$-вещи, необходимая для непротиворечивого выражения отношения содержательного следования, оказывается невыразимой в двухзначных исчислениях. Этим и объясняется безуспешность всех попыток адекватно отобразить отношение следования средствами двухзначных логик.

Основанная на алгебре совокупностей силлогистика оперирует общеутвердительными частноутвердительными, общеотрицательными и частноотрицательными посылками. В [6] показано, что отрицательные посылки могут быть сведены к утвердительным использованием наряду с утвердительными особенностями их антиподов. Возможность беспрепятственно использовать выражения алгебры совокупностей для представления всевозможных силлогистических посылок и моделирования достоверного рассуждения обеспечивается требованием сосуществования противоположностей, которое гарантирует неконстантность, содержательность первичных терминов. Согласно этому принципу, алгебраически формулируемому как

$$VxVx'VyVy'VzVz'\ldots \equiv 1,$$

термины представляют собой переменные, принимающие взаимоисключающие значения, что буквально соответствует гераклитову «все течет, все изменяется». В этом случае аристотелева силлогистика оказывается чрезвычайно простым и в то же время исчерпывающим исчислением, в котором силлогистический вывод осуществляется по естественным правилам.

При условии сосуществования противоположностей, Axy выражается одним, означающим несовместимость x и y', дизъюнктом $V'xy'$:

$$V'xy'\ (VxVx'VyVy') \equiv$$
$$\equiv V'xy'V(xy \vee xy')V(x'y \vee x'y')\ V(xy \vee x'y)\ V(xy' \vee x'y') \equiv$$
$$\equiv V'xy'Vxy\ Vx'y' \equiv Axy\ .$$

Силлогистический вывод производится совмещением совокупностей, выражающих задаваемые посылками отношения с учетом условия содержательности всех терминов. Например, доказательство модуса *barbara*:

$$AxyAyz \equiv (V'xy'VxyVx'y')(V'yz'VyzVy'z') \equiv$$
$$\equiv V'xy'V'yz'\ VxyVx'y'\ VyzVy'z' \equiv$$
$$\equiv V'xy'z\ V'xy'z'V'xyz'V'x'yz'VxyzVx'y'Vyz Vx'y'z' \equiv$$
$$\equiv V'(xy' \vee xz' \vee yz')\ VxyVx'y'\ VyzVy'z' \Rightarrow$$
$$\Rightarrow V'xz'VxzVx'z' \equiv Axz.$$

Выражения алгебры совокупностей для частных посылок (например, Ixy), а также не входящих в число посылок категорической силлогистики выражений $x \equiv y$ и $x \equiv y'$, аналогичным образом подчиняются принципу сосуществования противоположностей.

Алгебра совокупностей исчерпывающе компьютеризируется на основе троичного кода +,0,- представлением отношений 2^n-тритными шкалами, где n - число рассматриваемых терминов [7]. Компьютеризация категорической силлогистики достигается применением к шкалам операций пересечения $\bigcap$ и объединения $\bigcup$, а также преобразованием n-арной шкалы в $(n+1)$-арную и обратной ему элиминации термина. Например, общеутвердительная посылка Axy кодируется четверкой тритов +-0+, а доказательство модуса *barbara* $AxyAyz \Rightarrow Axz$ осуществляется путем пересечения представленных трехтерминными шкалами посылок и последующей элиминации среднего термина y [6]:

$$AxyAyz \equiv (+\text{-}0+)xy \bigcap (+\text{-}0+)yz \equiv (++\text{--}00++) \bigcap (+\text{-}0++\text{-}0+) \equiv$$
$$\equiv (+\text{---}0\text{-}0+)xyz \Rightarrow (+\text{-}0+)xz \equiv Axz.$$

Литература

1. Гильберт Д., Аккерман В. Основы теоретической логики. – М: ИЛ, 1947.

2. Лукасевич Я. Аристотелевская силлогистика с точки зрения современной формальной логики. - М.: ИЛ, 1959.

3. Стяжкин Н.И. Формирование математической логики. - М.: "Наука", 1967.

4. Н.П. Брусенцов. Парадоксы логики, здравый смысл и диалектический постулат Гераклита-Аристотеля. // «Программные системы и инструменты». Тематический сборник № 4. Под ред. Л.Н.Королева. – М. Издательский отдел ВМК МГУ, 2003. С. 35-38.

5. Аристотель. Сочинения в четырех томах. Том 2. – М: «Мысль», 1978.

6. Брусенцов Н.П. Реанимация аристотелевой силлогистики. // Реставрация логики. – М.: Фонд «Новое тысячелетие», 2005. С. 140-145.

7. Брусенцов Н.П., Владимирова Ю.С. Конструктная компьютеризация силлогистики. // «Математические методы распознавания образов: 13-я Всероссийская конференция». М.: МАКС-Пресс, 2007. С. 10-13.

Ельцин А.В.

Учебные материалы в интеллектуальной обучающей системе

1. Введение

Генетический метод обучения ГРОМ (Герменевтики и Развивающего обучения Мастер, [1]) использует в обучении концептуальную способность учащегося – понимание. В предмете пониманию служит синтезирующее знание: примеры, проблемы, понятия, доказательства, теоремы, теории. Поэтому разнообразны пути вхождения учащегося в смыслы предмета, а тем более предметов. Отсюда следуют решения по структуре авторского курса, используемой в системе для анализа учебного материала.

Во-первых, понятия (например, индекс книги) являются признаками выделяемых единиц текста и, по определению, считаются неупорядоченными. Также, по определению, особая роль у сложных понятий, как например, "частично-упорядоченная подполугруппа симметрической полугруппы монотонных преобразований упорядоченного множества". Они несут необходимую для метода обучения синтезирующую нагрузку, поэтому называются нами синтез-понятия. Синтез-понятие рассматривается как множество его образующих. Для анализа текста выделяются также метапонятия ("доказательство", "аксиома", "теорема", "следствие", "формула"). Нагруженная признаками фрагмент текста называется нами "СЕТ" – структурная единица текста.

Во-вторых, в системе проводится укрупнение единиц синтеза. СЕТы факторизуются по понятиям, поэтому далее рассматриваются как множества понятий. Далее рассматривается их представление через множества, поэтому за ними закрепляется то же имя – СЕТ. Сложность СЕТ определяется по суммарной мощности образующих понятий. Сложность применяется для автоматизации использования СЕТ в системе для обучения на основе линейного порядка. На СЕТах: определяем отношение сходства по образующим признакам

$$C_1 R C_2 \Leftrightarrow (C_1 \cap C_2) \neq \mathrm{O}$$

и тогда на графе связей СЕТ можем обнаружить новые метапризнаки (рассматривая в их качестве полные подграфы);
вводится метрика (Хэмминга):

$$\rho(C_1, C_2) = \left|(C_1 \cup C_2) \setminus (C_1 \cap C_2)\right|$$

В-третьих, полагаем, что синтезирующая разработанность в курсе понятия характеризуется весом понятия – числом СЕТ, содержащих вхождения понятия. Отсюда могут быть получены характеристики качества курса в отношении целей системы. На понятиях вводится отношение сходства

$$(p_1 R p_2) \Leftrightarrow \exists C(\{p_1, p_2\} \in C)$$

Оно позволяет, также обнаружить метапризнаки (полные подграфы), задействовать паразитические классы ([2]). На синтез-понятиях определяется расстояние

$$\rho(p_1, p_2) = |(\overline{p_1} \cup \overline{p_2}) \setminus (\overline{p_1} \cap \overline{p_2})|$$

В-четвертых, метапонятия и сложность СЕТ являются основанием для формирования каркасных единиц – выделяются примеры, проблемы, пример-проблемы. Им отводится роль объединяющего начала в отношении авторского учебного материала для концептуального тренажа учащегося. Пример – это поясняющий СЕТ, проблема – это фиксирующий СЕТ (например, теорема). Пример-проблема – это проблема, выраженная в доступной для учащегося форме примера. Пример-проблема является ключевой учебной единицей в системе. Они должны отыскиваться адаптивно к учащемуся на основе метамодели предметной области.

Таким образом, курс в системе имеет следующее многоуровневое представление:

$$K = \{\bigcup_i \pi_i + \bigcup_j CET_j + \bigcup_k p_k \mid P^{Asm}\},$$

где $\bigcup_i \pi_i$ – курс как концептуальная совокупность пример-проблем, $\bigcup_j CET_j$ – курс как синтезирующая совокупность СЕТ, $\bigcup_k p_k$ – логическая структура курса на совокупности понятий, P^{Asm} – предикаты, фиксирующие связность выделенных элементов курса.

2. Рабочее Место-1

На РМ1 готовятся учебные материалы для ИОС, в основном, из книг – авторских курсов. Они выстроены в иерархию, определенную координатизацией предметной области, представленной матрицей 3*3*3. В каждой точке матрицы может быть несколько курсов, поэтому число курсов измеряется сотнями. Отсюда следует необходимость создания быстрых и удобных средств для автоматизированной обработки курсов для включения их в учебную среду ОС.

Термин РМ1 указывает на то, что средства для обработки курсов должны быть не просто набором перекодировщиков из одного формата в другой, а средой для работы пользователя, поддерживающей

настройку используемых средств и корректировку исходных данных на всех этапах работы. Предполагается интеллектуальное участие человека, поэтому обеспечивается интерактивность – человек помогает системе там, где она не может справиться сама. А в тех случаях, когда возможны полностью автоматические стадии обработки, мы стараемся сделать их максимально быстрыми и прозрачными для пользователя.

В перспективе ИОС предполагается открытой для погружения новых курсов пользователем. Это повышает уровень требований – система должна с минимальным участием пользователя принять новый курс, проанализировать его и встроить в существующую иерархию курсов.

2.1. Подготовка учебных материалов для ИОС

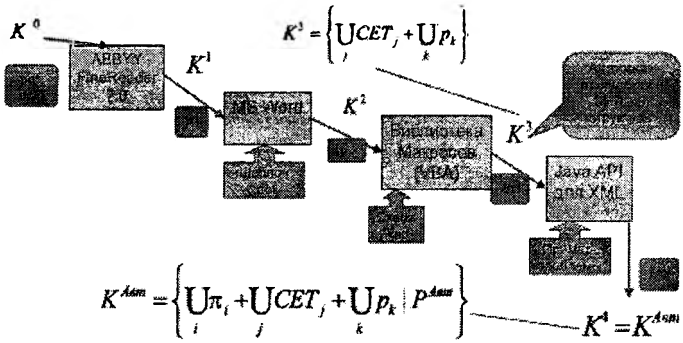


Рис.1. Схема обработки документа

На схеме показана цепочка преобразования документа для погружения его в систему. Результирующее представление документа K^{Asm} должно иметь логическую, синтезирующее СЕТовую и концептуальную структуры, отражающие авторское представление документа и обеспечивающие использование учебного материала для обучения в ИОС по методу ГРОМ. Формат результирующего документа задается специальной XML-схемой, которая определяет требования к документу на формальном уровне.

2.2. Интерактивная служба анализа

После обработки документов и получения их XML-представления необходимо проанализировать полученные результаты. Для анализа можно использовать различные программы, например, Stylus Studio которая поддерживает работу с XML-документами в трех

взаимосвязанных представлениях – текстовом, древовидном и табличном. Они позволяют увидеть структуру документа, выбрать нужную информацию из документа, например, средствами XPath – языка именования элементов документа XML. Тем не менее, возникает потребность в специализированном средстве, которое могло бы провести анализ полученного в результате обработки курса в отношении целей ИОС. Тогда важны следующие аспекты анализа:

- взаимодействие курсов в системе в отношении их связности для учебного действия;
- выделение в курсе пример-проблем; предлагается использовать кластеризацию; предполагается, что пример-проблемы попадут в центр кластеров документа;
- рассмотрение курса с точки зрения СЕТовой организации (разумно ли выделены, нет ли явных ошибок обработки);
- характеристика качества элементной структуры (понятия, которые не найдены ни разу; СЕТ, в которых не найдено вхождений понятий).

Все это требует создания специальных программных средств для анализа и визуализации результатов обработки курсов.

3. Реализация преобразования: $K^0 \Rightarrow K^{Авт}$

На пути преобразования в отношении перечисленных выше задач требуется создание специальных программных средств анализа и визуализации.

3.1. Распознавание текста - первичная обработка: преобразование $K^0 \Rightarrow K^1$

Первичная обработка документов проводится с использованием средств оптического распознавания текста. Для этой цели выбрана программа ABBYY FineReader 7.0, обладающая всеми необходимыми инструментами. Одним из удобных свойств этой программы является возможность сохранения результатов распознавания сразу в формате документа MS Word.

Наиболее распространенными форматами электронных книг в настоящее время являются форматы PDF и DJVU. Прямое копирование текста из них невозможно, так как в общем случае это графические форматы, хранящие тексты в виде специальным образом сжатых рисунков. Поэтому, первой задачей для помещения курсов в ИОС является преобразование документов этих форматов в текстовое представление при помощи средств оптического распознавания текста.

Формат PDF (Portable Document Format) достаточно просто переводится в текст средствами FineReader 7.0 – многостраничный файл при загрузке в систему оптического распознавания обрабатывается пакетно и результат обработки пакета может быть сохранен в одном файле формата MS Word.

Формат DJVU не распознается автоматически при помощи FineReader, но его можно при помощи различных программ, осуществляющих перевод между форматами, перевести в какой-нибудь формат, распознаваемый FineReader. Примером такой перекодировки в другой формат может служить преобразование документов через виртуальный принтер Universal Document Converter в формат PDF, либо перевод документов из формата DJVU в многостраничный формат TIFF с помощью программы DJVUDecode. Многостраничный формат TIFF также принимается в качестве входного формата для FineReader.

При применении FineReader для перевода электронных книг в текстовое представление необходимо следить за правильностью определения текстовых областей, картинок, колонтитулов и прочих частей документа. Настройки по умолчанию не всегда правильно решают эту задачу. Для аккуратного распознавания книги нужна ручная установка соответствующих блоков ("текст" – для текстовых областей, "картинка" – для формул и рисунков).

Выбранная схема первичной обработки документов позволяет помещать в систему даже те книги, которые не удалось найти в электронном варианте. Их нужно сначала отсканировать, а затем сохранить в один из форматов, принимаемых FineReader.

3.2. Общая авторская структура документа: преобразование $K^1 \Rightarrow K^2$

После распознавания и преобразования в формат MS Word, документ представляет собой однородный кусок текста, не имеющий никакой организации и, возможно, содержащий ошибки распознавания (например, неверно распознанные картинки). Устранение ошибок распознавания – малоинтересная ручная работа, но следует помнить, что без нее не обойтись. К сожалению, FineReader почти всегда неправильно распознает формулы, что является проблемой при обработке математических книг. Для ее решения необходима дополнительная ручная обработка, в перспективе предполагающая создание интерактивного средства.

Следующая задача, возникающая при работе с распознанным документом, – объединение ошибочно разбитых абзацев (они являются очень важной частью документа, так как на их основе готовятся СЕТ). Ошибки такого рода встречаются достаточно часто (например, когда абзац в исходной книге был разбит разрывом страницы), поэтому

разработан макрос, использующий эмпирическое правило – "абзац должен начинаться с большой буквы русского алфавита". Абзацы, не соответствующие этому правилу, макрос объединяет с предыдущими.

Главная обработка документов в среде MS Word – создание на исходном документе общей авторской структуры. Здесь требуется интеллектуальное участие человека для выделения в неструктурированном документе, полученном в результате распознавания, важных для использования в ИОС частей. Для этого используется специально разработанный шаблон, поддерживающий логическое, СЕТовое и пример-проблемное представление курса. Общая авторская структура документа задается вручную соответствующими маркерами-заголовками. Основная часть документа, из которой будет приготовлено СЕТовое представление курса, размечается заголовками "Сет_док" и "Конец_сет", индекс документа выделяется заголовками "Индекс_док" и "Конец_инд". Каждое понятие индекса ставится на отдельную строку. По итогам расстановки заголовков документ готов для обработки содержащейся в шаблоне макробиблиотекой. Заголовки – те ориентиры, которых придерживается макробиблиотека при работе с документом.

3.3. Уточнение авторской структуры (разметка XML-тэгами): преобразование $K^2 \Rightarrow K^3$

Выбор MS Word в качестве следующего звена в цепочке обработки документов неслучаен, так как кроме удобного универсального текстового редактора эта система содержит систему программирования на языке VBA ([3]), которая позволяет автоматизировать действия, выполняемые пользователем. Документ в среде MS Word имеет структуру коллекций объектов, что облегчает работу с ним как данными для объектно-ориентированного программирования.

Версии MS Word 2003 и выше поддерживают работу с XML-документами и сохранение документов формата MS Word в формате XML.

Для нас важны следующие операции:

1. подключение к документу XSD-схемы;
2. разметка документа XML-тэгами.

Так как любое действие пользователя системы MS Word может быть записано в виде макроса на языке VBA, эти операции также могут быть автоматизированы при помощи макросов.

Разметка документа XML-тэгами проводится при помощи макросов из библиотеки, содержащейся в шаблоне. Библиотека имеет графический интерфейс, повышающий эффективно использовать макросы, анализировать полученные изменения, при необходимости

"откатывать" документ до предыдущей версии, следить за тем, какие макросы уже отработали, а какие еще надо запустить для автоматизированного преобразования документа. Библиотеке обеспечена справочной системой, расположенной на сайте спецсеминара. При выборе макроса и нажатии на кнопку "помощь" открывается страница сайта, на которой содержится информация по работе с выбранным макросом и практические советы по применению макробиблиотеки.

После проведения всех подготовительных действий в системе MS Word, документ сохраняется в формат XML с условием "сохранить только текст" – в таком виде он готов к следующей стадии обработки.

3.4. Авторская структура (средствами Java API): преобразование $K^3 \Rightarrow K^{Авт}$

В процессе разработки инструментальных средств для PM1 выяснилось, что весь цикл обработки можно провести средствами макросов в среде MS Word. Однако работа с XML-документами в этой среде реализована не очень удобно (в частности, чтобы найти нужный элемент, надо перебрать все элементы), поэтому обработка XML-документов была перенесена на уровень Java API ([4]) для XML (используется библиотека DOM4j, [5]). Это позволило проводить трудоемкие операции быстрее, и упростить программный код, за счет использования технологии DOM (Document Object Model). DOM строит на основании XML-документа его древовидное представление в памяти, обеспечивая непосредственный доступ к любому элементу по его имени. Среди преимуществ библиотеки DOM4j можно отметить также возможность автоматического разбора (есть встроенный парсер документов) и автоматического сохранения дерева DOM в XML-документ.

Разделение обязанностей между программными средствами проведено по формату документа. Все, что делается на уровне текста и то, что требует пользовательской настройки на этом уровне, выполняется библиотекой макросов на языке VBA. А вся обработка, которая идет на уровне элементов и атрибутов XML, производится средствами DOM4j.

Для работы с XML-документами разработана программа "Рабочее место-1" (PM-1), которая строит по представлению K^3 представление K^4. Перенос трудоемких этапов обработки (таких как: поиск понятий из индекса в СЕТ; построение предикатов, показывающих связь между выделенными элементами курса) позволил повысить скорость обработки документов.

Программа "Рабочее место-1" обладает графическим интерфейсом для удобства пользователя. Вызов всех основных операций обработки документов производится нажатием кнопок на панели инструментов.

3.4.1 Разметка СЕТ понятиями из индекса

Создание СЕТов начинается с абзацев текста. Каждый абзац считаем заготовкой СЕТа. Макрос проходит по абзацам и размечает их XML-тэгами, выделяя текст, список понятий и числовой идентификатор СЕТа.

На следующей стадии обработки, которая проводится программой "Рабочее Место-1" (написанной на языке Java), в тексте СЕТ проводится поиск понятий по усовершенствованному алгоритму, учитывающему словоформы.

Работа со словоформами реализована через морфологический модуль RMU ([6]), с которым программа связывается через сетевое соединение (сокет). Программа РМ-1 посылает запрос на получение всех словоформ слова и получает ответ в виде списка словоформ. Для ускорения работы со словоформами разработан алгоритм кэширования словоформ – они сохраняются в файлах для быстрого обращения к ним. Также разработан интерфейс для редактирования словоформ, позволяющий корректировать уже сохраненные файлы кэша, а также дописывать словоформы для тех слов, которые неизвестны модулю.

После получения всех словоформ для всех понятий запускается процесс поиска. В тексте СЕТ для каждого понятия производится поиск всех слов (понятия зачастую многословны, например "Абсолютно целая алгебраическая функция"). Вычисляются координаты вхождений слов, то есть позиции в тексте СЕТ. Если каждое слово понятия входит в СЕТ хотя бы раз, то строится n-мерное декартово произведение множеств вхождений слов. Из этого декартова произведения выбираются отдельные вхождения – берется набор из n элементов, соответствующих вхождению всех слов понятия. Для всех таких наборов вычисляются медиана (средняя координат) и дисперсия (максимальное отклонение координат от медианы). Вхождение считается достоверным, если найден n-мерный набор, дисперсия которого не превышает половины количества слов понятия.

Вместо исходного индекса, который содержался в книге, может быть использован другой индекс, например, индекс интересующих нас понятий из предметной области или как-либо специально подготовленный для этой книги индекс (для того чтобы рассмотреть книгу не с точки зрения автора, а с точки зрения интересующих нас понятий). К сожалению, для некоторых книг индекса нет вообще, и его создание (на основе текста книги или индексов других книг) является отдельной задачей, решение которой представляется довольно сложным.

3.4.2 Подсчет статистики

Создан модуль, обеспечивающий подсчет статистики документа. Вычисляются следующие характеристики:

1. общее число СЕТ и понятий;
2. число вхождений каждого понятия в СЕТы;
3. число СЕТ, в которых на найдены вхождения понятий;
4. количество СЕТ, содержащие заданное число понятий.

Характеристики могут быть выражены как в абсолютных, так и в относительных величинах (в процентах). Такая статистика достаточна, чтобы: выявить ошибки обработки документа; выделить наиболее часто встречающиеся понятия; выяснить, какие СЕТ наиболее сильно нагружены понятиями (такие СЕТ важны для анализа курса на выявление пример-проблем).

На сегодняшний день обработано около 10 курсов. Анализ уже обработанных курсов показал, что в документах содержится от нескольких сотен до нескольких тысяч СЕТ и от 93 до 1578 понятий. Рекорд принадлежит книге Б. Ван Дер Вардена "Алгебра", содержащей 1578 понятий и 4554 СЕТ на 600 страниц. Найдено 221561 вхождение этих понятий в СЕТ. Этот пример дает представление о масштабе решаемых задач и позволяет прикинуть трудоемкость других операций анализа, например кластеризации СЕТ и выделения огрубленной структуры понятий.

Если объединить все индексы обработанных книг, то общее число понятий будет измеряться тысячами. В таком случае построение "грубой структуры" курса и выделение его основных узлов на основе анализа взаимоотношений курсов выходит на первый план по значимости.

3.4.3 Обработка формул и рисунков

Формулы и рисунки сохраняются в виде картинок и хранятся отдельно от документа. В XSD-схему документа введен тег "image", отвечающий за представление рисунков. Также в документе указывается путь к директории с формулами и рисунками.

FineReader выделяет рисунки в тексте недостаточно хорошо, и требуется ручная расстановка блоков с формулами и рисунками. Это достаточно трудоемкая задача, так как в математических книгах всегда много формул, но лучшего способа пока не придумано. Единственная программа, поддерживающая распознавание математических формул –InftyReader. Но, к сожалению, эта программа не поддерживает русский язык.

Итак, на этапе оптического распознавания формулы и рисунки оформляются в виде картинок, которые сохраняются при передаче

документа в MS Word. Написан специальный макрос, который проходит по документу, собирает все рисунки и складывает их в заданную директорию в формате BMP. На месте рисунков в тексте CET ставится тег “image”, в атрибутах которого прописывается путь к файлу с нужным изображением, ширина, высота и другие параметры изображения.

Литература

1. Громыко В.И.,... Обучающие системы «компьютерного» образования в высшей школе//Программные системы и инструменты. Труды ф-та ВМК, №7. М.: МГУ, 2005.
2. Шрейдер Ю. Б. Равенство, сходство, порядок. М.: Наука, 1971.
3. Гарнаев А. VBA. СПб.: BHV-Санкт-Петербург, 2005.
4. Мак-Лахлин Б. Java и XML. СПб.: Символ-плюс, 2002.
5. http://dom4j.org/
6. Проскурня М. Модуль морфологии русского языка. Спецификация SP-RMU 01-08. М.: МГУ, 2003.

Срджан Кадич, Предраг Станишич
Естественно-математический факультет
Университет Черногории

Алгоритм проверки сериализуемости выполнения множества транзакций

Многочисленные современные применения, такие как электронные запаси, транзакций денег, электронный бизнес по интернету, во многом зависят от программных технологий управление транзакциями.

Современные системы для управления базами данных должны обеспечить многофункциональную работу с базой данных. Мультипрограммные операционные системы поддерживают обработку пользовательских программ в условиях разделения процессорного времени и нескольких процессов. Появляется необходимость увеличения эффективности пользования ресурсами и в целом компьютерной системой.

Согласно потребностями для параллельного выполнения, интегрированная часть системы для управления базами данных есть механизм для управления транзакциями. Его задача состоит в том, чтобы обеспечить поддержку всем действиям, которые нужно предпринять, чтобы результаты параллельного выполнения множества транзакций были эквивалентны результатам их серийного выполнения. Тогда процесс проверки сериализуемости параллельного выполнения множества транзакций выполнит свою задачу. В этой работе предложен новый алгоритм проверки сериализуемости выполнения множества транзакций.

I ВВЕДЕНИЕ

I.1. Транзакция

Дефиниция: Транзакция это выполнение программы, которая представляет собой некоторую интеракцию в реальном окружении, и обеспечивает сохранение целостности базы данных в рамках модели окружения . Формально, транзакцию Ti можно представить как ряд пар:

$$T_i = ((O_{i1}, X_{i1}), \ldots, (O_{in}, X_{in}))$$

причем: O_i - операция, Xi - данные над которыми выполняются операции.

Выполнение транзакции Ti в значении I (Ti), можно представить как ряд троек: $I(T_i) = ((O_{i1}, X_{i1}, T_i), ..., (O_{in}, X_{in}, T_i))$.

I.2. Выполнение множества транзакции

Предположим, что в мультипрограммной компьютерной системе существует множество n активных транзакций, $T = \{T_1, ..., T_n\}$. Транзакции из этого множества могут выполнятся :

- Серийно – сначала выполняется полностью одна транзакция, потом другая, потом третья и так подряд пока не выполнятся все транзакции из множества T.

- Параллельно (несерийно) – прежде чем полностью выполнится одна транзакция, стартует другая, перед тем как выполнится полностью другая, стартует какая то третья транзакция, или, продолжает свою работу первая, и так одна сменяет другую, пока все транзакции не выполнятся полностью.

В зависимости от того в какой последовательности сериальной или параллельной, производится выполнение зависит одинаковый или нет получится результат, а так же получает ли параллельная последовательность одинаковый результат, как и несериальная последовательность. Введем следующие дефиниции последовательностей:

- эквивалентная последовательность
- сериальная последовательность.

Дефиниция: Выполнение множества транзакции $T = \{T_1, ..., T_n\}$ в значении I(T), можно представить как ряд троек: $I(T) = ((O_{i1}, X_{i1}, T_{I1}), ..., (O_{ir}, X_{ir}, T_{Ir}))$ где: O_i – операция, X_i – данные над которыми выполняется операция, T_i – транзакции из множества T, $r - r = m_1 + ... + m_n$, m_i – номер паров (O,X) транзакции Ti.

Будем пользоваться обозначениями: $INDSET(T_i) = \{i_p \mid T_{ip} = T_i\}$ и $I_{INDSET(Ti)} = T_i$, i = 1, ..., n.

I.3. Параллельное выполнение множества транзакции

Параллельная последовательность выполнения транзакции из множества $T = \{T_1,..., T_n\}$ есть последовательность выполнения в которой ряд инструкций одной транзакции перерываются рядом инструкций другой транзакции.

Дефиниция: Параллельная последовательность выполнения множества транзакции $T = \{T_1, ..., T_n\}$ это ряд троек : $I(T) = ((O_{i1}, X_{i1},$

T_{l1}), …, (O_{ir}, X_{ir}, T_{lr})), что означает : INDSET(T_i) = {i_p ¦ T_{ip} = T_i}, $I_{INDSET\ (Ti)}$ = T_i, i = 1, ..., n и INDSET (T_i) не интервал, i = 1, …, n.

Метка времени в которой выполняются операции транзакции T_i не дают континуального времени интервала как это происходит в случае с сериальным выполнением множества транзакции.

Число различных способов, путем которых можно параллельно выполнить множество от n транзакций T={T_1, …, T_n} представлен следующей формулой:

$$(m_1! + ... + m_n!) / (m_1! * ... * m_n!)$$

Для конкретных последовательностей выполнения транзакции существуют следующие факты:

- Любое параллельное выполнение транзакции не обязательно обеспечивает сохранение устойчивого состояния базы данных.

- Состояние некоторых частей базы данных зависит от параллельной последовательности выполнения транзакции.

- Существует параллельное выполнение транзакции, которые обеспечивает сохранение устойчивого состояния базы данных.

- Существует параллельное выполнение транзакции, которое дает такой же результат как некоторая сериальная последовательность.

I.4. Эквивалентное и сериальное выполнение множества транзакции

Дефиниция: Две последовательности выполнения I_1 и I_2, множества транзакции T^1 и T^2, представленное как:

$$I_1 (T^1) = ((O_{i1}, X_{i1}, T_{l1}), …, (O_{ir}, X_{ir}, T_{lr}))\ ,$$

$$I_2 (T^2) = ((O_{j1}, X_{j1}, T_{h1}), …, (O_{jr}, X_{jr}, T_{hs}))\ ,$$

являются эквивалентными последовательностями, что будем обозначать I1 = I2, если выполнены два необходимых и достаточных условия:

1. Множества транзакций, которые выполняются по последовательностям

$I_1 (T^1)$ и $I_2 (T^2)$ - одинаковы $T^1 \equiv \{T_1, …, T_n\} \equiv T^2$, и

2. Если транзакция T_i в последовательности I_1 читает или вписывает данные X, и перед этим поменяла транзакцию T_j, это должно отразиться и в последовательности I_2.

Формально: Для каждого X и каждую пары упорядоченных троек (O_{ik}, X_{ik}, T_{lk}), (O_{if}, X_{if}, T_{lf}) из I_1 где

- $X_{ik} \equiv X_{if}$,
- $O_{ik} \equiv$ Write ,
- $O_{if} \equiv$ Read or Write ,
- $T_{lk} \neq T_{lf}$,

- $k < f$,

существует пара упорядоченных троек (O_{jk}, X_{jk}, T_{hk}), (O_{jf}, X_{jf}, T_{hf}) из I_2 где

- $X_{jk} \equiv X_{jf} \equiv X_{ik} \equiv X_{if}$,
- $O_{jk} \equiv O_{ik}$,
- $O_{jf} \equiv O_{if}$,
- $T_{hk} \equiv T_{lk}$,
- $T_{hf} \equiv T_{lf}$,
- $k < f$.

Дефиниция. Сериализуемая последовательность выполнения множества транзакций это та параллельная последовательность, которая эквивалентна некоторой сериальной последовательности.

II АЛГОРИТАМ ПРОВЕРКИ СЕРИАЛИЗУЕМОСТИ

Для проверки сериализуемости последовательности выполнения множества транзакции конструируется граф последовательности.

Дефиниция. Помеченный граф последовательности – это направленный граф $G^{marked}_{sequence} = (T, R)$, в котором

- T, множество вершин графа (их число равно числу транзакций),
- $R = \{R_{wr}, R_{rw}, R_{ww}\}$, множество граней графа
- R_{wr}, грань записи- чтения
- R_{rw}, грань чтение – записи
- R_{ww}, грана записи-записи.

Множество вершин T подходит множеству транзакций которые выполняются по некоторой последовательности. Направления граней $\{R_{wr}, R_{rw}, R_{ww}\}$ соответствуют последовательностями транзакций, которые выполняют указанные операции для каждого данного.

II.1 Процедура конструкции помеченного графа последовательности

1. Каждой транзакции из множества $T = \{T_1, \ldots, T_n\}$, которые выполняются по определенной последовательности I(T), присваивается по одной вершине помеченном графе последовательности.

2. Для каждого данного из множества $X = \{X_1, \ldots, X_m\}$, для каждой пары упорядоченных троек (O_{ik}, X_{ik}, T_{lk}), (O_{if}, X_{if}, T_{lf}) из I (T), где:

- $X_{ik} \equiv X_{if}$,
- $O_{ik} \equiv$ Write,

- $O_{if} \equiv$ Read ,
- $k < f$,
- $T_{lk} \neq T_{lf}$,

конструируется w_r грань от вершины T_{lk} до вершины T_{lf} в помеченном графе последовательности.

3. Для каждого данного из множества $X = \{X_1, \ldots, X_m\}$, для каждой пары упорядоченных троек (O_{ik}, X_{ik}, T_{lk}), (O_{if}, X_{if}, T_{lf}) из I (T), где:

- $X_{ik} \equiv X_{if}$,
- $O_{ik} \equiv$ Read ,
- $O_{if} \equiv$ Write ,
- $k < f$,
- $T_{lk} \neq T_{lf}$,

конструируется w_r грань от вершины T_{lk} до вершины T_{lf} в помеченном графе последовательности.

4. Для каждого данного из множества $X = \{X_1, \ldots, X_m\}$, для каждой пары упорядоченных троек (O_{ik}, X_{ik}, T_{lk}), (O_{if}, X_{if}, T_{lf}) из I (T), где

- $X_{ik} \equiv X_{if}$,
- $O_{ik} \equiv$ Write ,
- $O_{if} \equiv$ Write ,
- $k < f$,
- $T_{lk} \neq T_{lf}$,

конструируется w_w грань от вершины T_{lk} до вершины T_{lf} в помеченном графе последовательности.

Помеченный граф последовательности конструируется с множеством транзакций, которые не обязательно читают данные прежде выписывания (записи).

Алгоритм проверки сериализуемости выполнения множества транзакции состоит из двух шагов:

1. Редукция помеченного графа последовательности.
2. Обнаружение цикла в редуцированном помеченном графе последовательности.

II.2 Редукция помеченного графа последовательности

В помеченном графе последовательности можно уменьшить число граней, что уменьшает комплексность графа. Уменьшение можно сделать удалением граней, которые представляют пустые записи одного данного.

В процедуру редукций помеченного графа последовательности входят следующие операции.

a) Из помеченного графа последовательности убираются w_w грани, которые не подходят к гранам w_r и r_w и которые заканчиваются в некоторой вершине T_{If}, и продолжаются выходной граню w_w, которая не совпадает с w_r граню. Существование такой грани означает, что транзакция T_{If} выполняется запись без пользы.

b) Из помеченного графа последовательности убираются w_r и r_w грани, которые заканчиваются в вершине транзакции T_{If}, которая выполняет запись без пользы.

II.3 Обнаружение цикла в редуцированном помеченном графе последовательности

В этом шаге проверяется существование цикла в редуцированном помеченном графе последовательности. Если существует цикл, тогда рассматриваемая последовательность несериализуема. Для обнаружения цикла в редуцированном помеченном графе последовательности можно использовать следующий алгоритм, обоснованный на топологической сортировке графов.

```
Procedure TopSort
Queue q(NUM_VERTICES);
int counter = 0;
Vertex v, w;
q.Empty();          'Инициализация последовательности
for each vertex v
        if (v.indegree == 0) then q.endqueue(v);
while (!q.IsEmpty())
{
        v=q.dequeue();
        counter++;
        for each w adjacent to v
                if (--w.indegree == 0) then q.endqueue(w);
}
if (counter != NUM_VERTICES) then CycleFound();
End procedure
```

Пример 1. Последовательность выполнения множества транзакции T = $\{T_1, T_2, T_3\}$ приведена в таблице 1. Применением предложенного алгоритма проверим сериализуемость этой последовательности.

Время	T_1	T_2	T_3
t_1	Read (Q)		
t_2		Write (Q)	
t_3	Write (Q)		
t_4			Write (Q)

Таблица 1

1. Конструкция Помеченного графа последовательности
2. Согласно предложенному алгоритму - конструируется помеченный граф последовательности. В этот процесс входят следующие шаги:

1.1 Каждой транзакции множества транзакции T = {T1, T2, T3} присваивается по одной вершине в помеченном графе последовательности.

1.2 Для всех данных из множества X = {Q} проверяем существование пар упорядоченных троек [(Write, Q, T_{lk}), (Read, Q, T_{lf})] из последовательности выполнения I(T_1, T_2, T_3). В этом случае не существует ни одной такой пары т.е. в помеченном графе последовательности не существует грани типа w_r.

1.3 Для данных из множества X = {Q} проверяем существование пар упорядоченных троек [(Read, Q, T_{lk}), (Write, Q, T_{lf})] из последовательности выполнения I(T_1, T_2, T_3). В этом случае существуют две таких пары т.е. [(Read, Q, T_1), (Write, Q, T_2)] и [(Read, Q, T_1), (Write, Q, T_3)]. В помеченном графе последовательности существуют две грани типа r_w ($T_1 \rightarrow T_2$ и $T_1 \rightarrow T_3$).

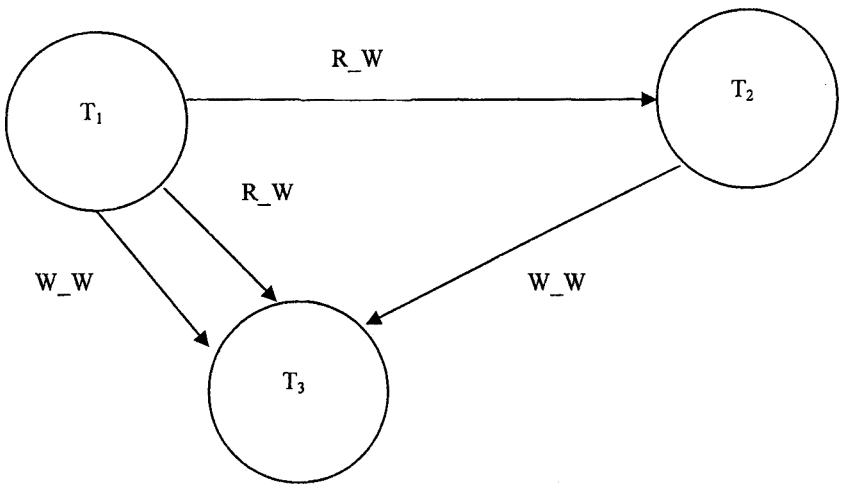
1.4 Для данных из множества X = {Q} проверяем существование пар упорядоченных троек [(Write, Q, T_{lk}), (Write, Q, T_{lf})] из последовательности выполнения I (T_1, T_2, T_3). В этом случае существуют две таких пари т.е. [(Write, Q, T_1), (Write, Q, T_3)] , [(Write, Q, T_2), (Write, Q, T_1)] и [(Write, Q, T_2), (Write, Q, T_3)]. В помеченном графе последовательности существуют три грани типа w_w ($T_1 \rightarrow T_3$, $T_2 \rightarrow T_1$ і $T_2 \rightarrow T_3$).

3. Редукция помеченного графа последовательности. В этот процесс входят следующие шаги:

2.1 Из помеченного графа последовательности убирается грана w_w ($T_2 \rightarrow T_1$), потому что не входит в w_r и r_w граням и не заканчивается в вершине T1, но продолжается выходящей граню w_w ($T_1 \rightarrow T_3$), которая не входит в w_r гране. Существование такой w_w грани означает то, что транзакция T1 выполняет запись без пользы.

2.2 Из помеченного графа последовательности убирается w_r и r_w грани, которые заканчиваются в вершине T1. В этом случае нет таких w_r и r_w граней.

2.3 Пока в помеченном графе последовательности существует w_w грань, которую нужно убирать, повторяются шаги 2.1 и 2.2. Когда таких вершин больше нет, процесс редукции помеченного графа последовательности завершен. Полученный граф это редуцированный помеченный граф последовательности.



Редуцированный помеченный граф последовательности

3. Обнаружение цикла в Редактированной Помеченного графа последовательности. Согласно предложенному алгоритму проверяется существование цикла в ориентированном графе. Редактированный Помеченный граф последовательности становится обыкновенным ориентированным графом (одна грань от вершины T_i до вершины T_j). Применяем указанный метод топологической сортировки ориентированного графа. В обнаружение цикла входят следующие шаги:

3.1 Инициализация – ряд (структура данных) получает статус пустой. Counter (счетчик посещаемых вершин ориентированной графи) инициализируется значением 0.

3.2 Для каждой вершины в ориентированном графе считается число входящих граней (входящая степень вершины). В этом примере $T_1(0)$, $T_2(1)$ и $T_3(2)$. Согласно алгоритму вершина T_1 вставляется в конец ряда. Ряд = { T_1}.

3.3 Если Ряд не пустой, берется вершина с начала ряда (V = T_1). Увеличиваем counter на 1 (число посещаемых вершин).

3.4 Убираем все грани от вершины V до вершины W т.е. уменьшаем входные степени соседних вершин. В этом случае $T_2(1) \rightarrow T_2(0)$, Ряд = Ряд /{ T_2} и $T_3(2) \rightarrow T_3(1)$.

3.5 Повторяем процесс пока ряд не пустой. Берем вершину с начала РЯДА (V = T2). Увеличиваем counter на 1 (counter = 2, Ряд = {}).

3.6 Убираем все грани от вершины v до вершины w т.е. уменьшаем входные степени соседних вершин. В этом случае $T_3(1) \rightarrow T_3(0)$, Ряд = Ряд / { T_3}.

3.7 Повторяем пока ряд не пустой. Берем номер вершины с начала Ряда (V = T3). Увеличиваем counter на 1 (counter = 3, Ряд = { }).

3.8 Убираем все грани от вершины V до вершины W т.е. уменьшаем входные степени соседних вершин. В этом примере нет соседних вершин.

3.9 Ряд пустой, сравниваем значение counter и число вершин ориентированного графа. В данном случае они имеют одинаковое значение, что означает, что не существует цикла в ориентированном графе, т.е. не существует цикла в редуцированном помеченном графе последовательности. Следует, что рассматриваемая последовательность выполнения I(T) сериализуемая.

III Заключение

Интегрирование в системы для управления базами данных механизма для управления транзакциями должна обеспечивать поддержку и всех активностей, которые необходимы для того, чтобы результаты параллельного выполнения множества транзакций были эквивалентны результатам их серийного выполнения. Процесс проверки сериализуемости параллельного выполнения множества транзакции имеет важное значение. В этой работе предложен новый алгоритм проверки сериализуемосты выполнения множества транзакций, которые не обязательно читают данные до их считывания. Предложенный алгоритм позволяет увеличить возможности механизма для управления транзакциями.

Литература

[1] Garcia-Molina H., Ullman J., Widom J., *Database System Implementation*, Prentice Hall, 2000

[2] Elmasri R., Navathe S., *Fundamentals of Database Systems*, The Benjamin/Cummings Publishing Company Inc., 1994

[3] Korth H., Silbershatz A., *Database System Concepts*, McGraw-Hill, 1988

[4] Gray J., Reuter A., *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann ,1992

[5] Ullman J., *Principles of Database Systems*, Computer Science Press, 1982

[6] Bernstein P, Hadzilacos V., Goodman N., *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987

[7] Bernstein P, Newcomer E., *Principles of Transaction Processing*, Morgan Kaufman , 1997

Раздел V.

Инструментальные средства

Балашов В.В., Бахмуров А.Г., Волканов Д.Ю., Смелянский Р.Л., Чистолинов М.В., Ющенко Н.В.

Применение стенда полунатурного моделирования для разработки вычислительных систем морского навигационного комплекса

Введение

Современные бортовые встроенные системы реального времени (ВС РВ) представляют собой многомашинные вычислительные комплексы. Количество каналов связи между приборами в составе ВС РВ достигает нескольких десятков. На этапе комплексирования ВС РВ возникает ряд задач, требующих инструментальной поддержки, в том числе:

- проверка соответствия приборов ВС РВ требованиям технического задания, в том числе в части приёма и передачи данных по внешним интерфейсам;
- отработка взаимодействия между приборами ВС РВ по бортовым каналам передачи данных;
- комплексное тестирование и отладка ПО ВС РВ, в том числе ПО, выполняемого распределено на различных приборах;
- оценка надёжности архитектуры ВС РВ, в том числе наличия резерва пропускной способности каналов передачи данных и устойчивость аппаратно-программных средств ВС РВ к сбоям при передаче данных;
- построение расписаний обмена данными по бортовым каналам, а также проверка правильности отработки этого расписания приборами в составе ВС РВ.

Разработка приборов, входящих в состав ВС РВ, на практике происходит распределённо и выполняется различными организациями. Готовность различных приборов к комплексированию наступает в разные моменты времени. Для соблюдения сроков комплексирования ВС РВ необходимо начинать работы по комплексирования с неполным комплектом доступных приборов. В данной работе описаны

программно-аппаратные средства (стенд) полунатурного моделирования бортовых ВС РВ, позволяющие осуществлять комплексирование ВС РВ и решать перечисленные выше задачи 1-5 поэтапно, расширяя состав стыкуемых приборов по мере их готовности в виде натурных образцов.

Стенд разработан в лаборатории вычислительных комплексов (ЛВК) факультета ВМК МГУ[1]. Особенностью стенда является возможность сравнительной оценки архитектур ВС РВ при помощи моделирования на ранних этапах разработки ВС РВ, что отличает стенд от существующих решений, в частности, от описанного в работе [2].

1. Задачи стенда полунатурного моделирования

Основными задачами стенда полунатурного моделирования (ПНМ) являются:

- проведение исследований и оценка технических решений в области структуры ВС РВ, характеристик приборов и их программного обеспечения;
- комплексная отработка оборудования при решении задач информационного взаимодействия и прикладных задач ВС РВ;
- проверка работоспособности аппаратуры, в т.ч. проверка соответствия функционирования приборов и каналов передачи данных требованиям технического задания.

Основными методами исследования ВС РВ в стенде ПНМ являются имитационное и полунатурное моделирование. В зависимости от степени готовности приборов ВС РВ, в составе стенда могут использоваться полностью программные модели всех приборов, либо комплекс из натурных образцов приборов и программных моделей приборов, собранных в единый стенд и сопряжённых через аппаратные каналы бортовых интерфейсов.

Также может варьироваться уровень детальности моделирования - от так называемых "интервальных" моделей, которые отрабатывают заданные циклограммы обменов, не выполняя вычисление передаваемых прибором данных, до полных функциональных моделей, эквивалентных реальным приборам по составу и значениям выдаваемых в бортовые каналы данных и включающих в свой состав реальное ПО приборов.

Средства разработки моделей, входящие в стенд ПНМ, позволяют быстро создавать модели, имитирующие "окружение" отлаживаемого прибора и взаимодействующие с ним через аппаратные каналы передачи данных.

2. Структура программных и аппаратных средств стенда

ПО стенда ПНМ включает в себя набор программных инструментов, обеспечивающих решение следующих задач, связанных с моделированием:

- создание имитационных моделей приборов ВС РВ, а также вспомогательных моделей (например, модели внешней среды);
- организация взаимодействия моделей, выполнения набора моделей, их взаимодействия с аппаратурой в модельном и в реальном времени по бортовым каналам с возможностью внесения отказов в каналы;
- управление процессом моделирования в диалоговом режиме, либо выполнение автономного эксперимента без участия оператора;
- оперативное отображение результатов моделирования в графическом и табличном виде;
- регистрация и обработка результатов моделирования, в том числе взаимодействие с аппаратными мониторами каналов передачи данных.

ПО стенда функционирует под управлением ОС Debian GNU/Linux. Оборудование стенда ПНМ включает в себя следующие компоненты (Рис. 1):

- инструментальные машины частных моделей (ИМ ЧМ);
- инструментальные машины-регистраторы обмена по бортовым каналам передачи данных (ИМ-регистраторы);
- машины АРМ инженера-экспериментатора;
- натурные приборы из состава ВС РВ;
- натурные каналы информационного обмена, к которым подключены приборы, ИМ ЧМ и ИМ-регистраторы.

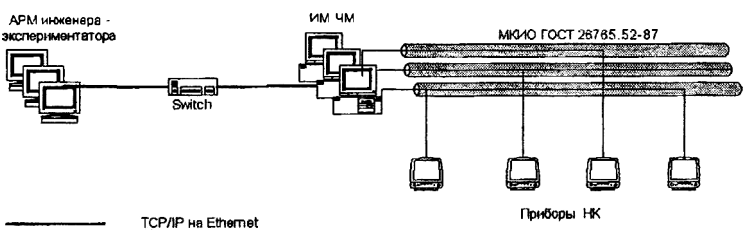


Рис. 1. Структура стенда ПНМ

Инструментальные машины и АРМ представляют собой персональные компьютеры, объединённые сетью Ethernet. На ИМ ЧМ осуществляется имитационное моделирование приборов с помощью разработанных исследователем моделей. При моделировании соблюдается соответствие модельного и реального (астрономического) времени с погрешностью порядка 0,0001 секунды. Указанная точность обеспечивается применением на ИМ ЧМ специализированных расширений реального времени для ядра ОС Linux, входящих в среду выполнения моделей [1]. В состав ИМ ЧМ входят адаптеры бортовых каналов передачи данных (поддерживаются стандарты МКИО (ГОСТ Р 52070-2003) [3], ARINC-429 (ГОСТ 18977-79, РТМ 1495-75), частично поддерживается стандарт Fibre Channel. При помощи этих адаптеров модели устройств ВС РВ осуществляют обмен по каналам в реальном времени, полностью или в заданной части имитируя работу реальных устройств. Необходимо отметить, что на канале МКИО модель устройства может выполнять роль контроллера канала, что обеспечивает возможность комплексирования подмножества устройств ВС РВ, не включающего устройство-контроллер канала.

Реализованная в стенде функциональность взаимодействия моделей устройств ВС РВ и натурных устройств в реальном времени позволяет обеспечить пошаговое комплексирование ВС РВ с поэтапной заменой программных моделей устройств на натурные устройства. Таким образом. комплексное тестирование и отработка взаимодействия могут выполняться на произвольном сочетании натурных и моделируемых устройств.

ИМ-регистраторы осуществляют мониторинг информационного обмена по бортовым каналам и запись протоколов регистрации на жёсткий диск для последующего анализа. При выполнении эксперимента, на всех инструментальных машинах поддерживается единый отсчёт времени, расхождение счётчиков времени на разных машинах не превышает 0,00001 секунды. Для генерации строго одновременных событий в целях синхронизации времени применяется специализированный протокол, использующий соединение LPT-портов ИМ ЧМ и ИМ-регистраторов. Периодически одна из ИМ ЧМ (мастер синхронизации времени) посылает сигнал через свой порт LPT. Остальные машины принимают сигнал и фиксируют одновременное событие. Регистрация событий в моделях (на ИМ ЧМ) и регистрация обмена по каналам (на ИМ-регистраторах) выполняется в едином времени, что позволяет с высокой точностью отображать на единой оси времени события на разных каналах и в разных моделях.

Средства АРМ инженера-экспериментатора обеспечивают решение следующих задач:

- разработка моделей с поддержкой управления версиями моделей;
- настройка эксперимента, в т.ч. распределение моделей по инструментальным машинам и настройка регистрации по каналам;
- управление экспериментом с поддержкой оперативного изменения значений параметров моделей с консоли оператора;
- оперативное наблюдение за ходом эксперимента, включая отображение значений параметров моделей и отображение результатов регистрации;
- анализ результатов эксперимента, в том числе трасс значений параметров моделей, трасс событий в моделях и каналах.

Для разработки моделей используется специализированный язык описания моделей, представляющий собой подмножество языка Си, расширенное средствами привязки работы модели к реальному времени и средствами организации межмодельного взаимодействия.

3. Технология комплексирования ВС РВ с применением стенда

При комплексировании ВС РВ с применением стенда ПНМ, средства стенда применяются совместно со средствами инструментальной системы планирования обменов по каналу МКИО (САПР циклограмм) [4, 5], также разработанное в ЛВК. В составе САПР циклограмм следует выделить следующие компоненты, существенные для совместного применения САПР и стенда:

- база данных (БД) проекта ВС РВ, содержащая информацию о структуре ВС РВ (состав приборов и каналов передачи данных, структура бортовой сети) и о порядке обмена между приборами (входные и выходные данные устройств, структура передаваемых сообщений, расписание информационного обмена);
- средства автоматического построения расписания информационного обмена для выполнения на устройстве-контроллере канала MIL STD-1553B.

В рамках разработанной технологии, процедура комплексирования ВС РВ в условиях доступности заданного (как правило, неполного) набора аппаратных устройств, включает следующие шаги:

1. Подготовка аппаратного обеспечения стенда, в том числе подключение ИМ ЧМ, ИМ-регистраторов и натурных приборов к каналам передачи данных.
2. Ввод в БД проекта информации о структуре ВС РВ и порядке информационного обмена между приборами.
3. Автоматическое формирование интерфейсной части моделей приборов ВС РВ (входные и выходные параметры моделей, описание интерфейсов моделей с бортовыми каналами, описание связей между моделями).
4. Реализация логики функционирования моделей тех устройств, которые отсутствуют в аппаратном исполнении.
5. Автоматическое формирование (средствами САПР циклограмм) расписаний обмена по каналам MIL STD-1553B.
6. Автоматическое формирование кода задания расписания: для аппаратных устройств-контроллеров канала в составе ВС РВ и для моделей устройств-контроллеров канала.
7. Автоматическое формирование кода упаковки и распаковки передаваемых по каналам сообщений для аппаратных устройств в составе ВС РВ и для моделей устройств.
8. Настройка эксперимента, включающее распределение моделей по ИМ ЧМ и задание настроек регистрации обмена по бортовым каналам и событий в моделях.
9. Выполнение эксперимента, при котором в составе стенда одновременно работают натурные приборы ВС РВ и модели отсутствующих приборов.
10. Анализ результатов эксперимента, в том числе автоматизированная проверка зарегистрированных протоколов обмена по бортовым каналам на соответствие эталонным расписаниям обменов, заложенным в БД проекта.

При пополнении состава доступных натурных устройств ВС РВ, комплексирование ВС РВ в расширенном составе требует выполнения только тех шагов, которые связаны с подключением новых устройств, проведением экспериментов с их участием и анализа результатов этих экспериментов. При комплексировании последующих версий ВС РВ (например, в рамках создания линейки ВС РВ), разработанные модели, наполнение БД и настройки экспериментов могут быть повторно использованы.

Стенд ПНМ может быть также применятся при отработке функционального ПО (ФПО) и системного ПО (СПО) приборов ВС РВ с использованием моделей этих приборов. Для этого в составе программных средств стенда должны быть реализованы модели низкоуровневых сервисов ОС прибора, в частности - модели сервисов межпроцессного взаимодействия и модели драйверов, работающих с адаптерами бортовых интерфейсов. После реализации моделей

указанных сервисов, в состав моделей устройств ВС РВ могут быть включены исходные тексты разрабатываемого ПО. Выполнение моделирования с задействованием исходных текстов ПО позволяет осуществлять тестирование и отработку ПО без задействования натурного прибора, который может быть недоступен в аппаратном исполнении и выполнять детальное наблюдение за изменением выбранных переменных ПО с задействованием механизма оперативного отображения параметров моделей.

По завершении отработки ПО на модели, выполняется включение этого ПО в состав реального прибора (по факту готовности прибора) и продолжается отработка ПО уже в составе прибора.

В случае если прибор доступен в аппаратном исполнении с самого начала процесса комплексирования ВС РВ, использование стенда позволяет расширить состав ПО, задействуемого при отработке прибора, за счёт подачи на интерфейсы прибора данных по каналам бортовых интерфейсов и активизации ПО, отвечающего за приём и передачу данных по этим каналам. Частным случаем такого подхода к отработке ПО на натурном приборе является моделирование всего окружения прибора и анализ ответов прибора на тестовые наборы данных, формируемые моделями и подаваемые на вход прибора.

Стенд ПНМ имеет открытую архитектуру, позволяющую выполнять интеграцию со стендами других разработчиков. Подобная интеграция целесообразна в случаях, когда для выполнения некоторых задач отработки ВС РВ необходимо использование спецвычислителей в составе стенда. Примерами таких задач могут служить динамическое формирование и регистрация видеоданных в реальном времени, а также получение управляющих воздействий от пилота через органы управления в стенде-имитаторе кабины пилота. Существующая реализация стенда ПНМ задействует для интеграции с внешними стендами специализированный протокол обмена, основанный на технологии Ethernet. В перспективе рассматривается возможность применения технологии HLA[6].

4. Применение стенда для оценки архитектуры ВС РВ морских навигационных комплексов

На ранних этапах разработки ВС РВ необходимо решить задачу выбора архитектуры ВС РВ. Как правило, конкретная архитектура выбирается из нескольких альтернатив. В связи с необходимостью определения архитектуры на раннем этапе разработки ВС РВ, нет возможности осуществлять натурные испытания различных вариантов архитектуры для определения наилучшего из них.

Средства моделирования в составе стенда ПНМ поддерживают создание полностью программной модели ВС РВ. В состав среды

моделирования входят модели бортовых каналов передачи данных, которые позволяют выполнять моделирование без задействования аппаратных каналов. В состав моделей приборов ВС РВ может быть включено натурное ПО этих приборов.

Средства стенда ПНМ позволяют решать следующие задачи оценки архитектуры ВС РВ:

- оценка резерва пропускной способности каналов передачи данных посредством моделирования обмена по каналу в соответствии с реальным расписанием обмена;
- исследование устойчивости информационного обмена к помехам на канале за счёт задействования функциональности модели канала по имитации отказов на канале (искажение разрядов в сообщениях, потеря сообщений и т.п.);
- оценка загрузки центральных процессоров приборов-вычислителей в составе ВС РВ;
- оценка отказоустойчивости ВС РВ, применяя метод анализа механизмов отказоустойчивости, основанный на имитационном моделировании [7].

При моделировании морских навигационных комплексов (НК) был выявлен ряд особенностей, которые потребовали доработки средств стенда:

- наличие приборов, основной функцией которых является ретрансляция данных между сегментами бортовой сети;
- необходимость отработки ошибок передачи данных в каналах МКИО и отработка реакции на них в СПО приборов;
- использование адаптеров каналов МКИО в режиме монитора канала и переключение в динамике между режимами монитора канала и контроллера канала при реконфигурации приборов;
- потребность использования при разработке новых приборов существующих исходных текстов СПО аналогичных приборов;
- необходимость проведения длительных (до 10 суток) экспериментов, вследствие особенностей накопления и проявления ошибок.

В средства стенда ПНМ был внесён ряд изменений. Перечислим и рассмотрим их подробно.

1. Добавлена поддержка использования в моделях приборов НК фрагментов исходных текстов ПО приборов.

При создании полных функциональных моделей приборов морского НК, с воспроизведением всей логики вычислений, выполняемых в приборе, необходимо использовать исходные тексты ПО моделируемого прибора или аналогичных приборов. Это позволяет сократить время разработки моделей, избежать ошибок при написании

алгоритмов ПО в стенде ПНМ заново, так как предыдущая версия ПО уже прошла тестирование и верификацию.

2. *В ПО стенда добавлена поддержка в адаптерах МКИО режима монитора канала и возможности оперативного переключения между режимами монитора, контроллера и оконечного устройства;*

При обменах по каналу МКИО важную роль играет монитор канала. Монитор канала – это интерфейс к каналу, который не передаёт информацию, но отслеживает все передаваемые сообщения и делает их доступными для своего абонента. В резервируемых приборах НК адаптер МКИО при реконфигурации прибора может динамически переключаться из режима контроллера канала (в основном приборе) в режим монитора (в резервном приборе). Чтобы моделировать описанное переключение режимов, в стенд добавлена новая возможность динамического переключения режимов интерфейса канала МКИО.

3. *Реализована поддержка генератора потока отказов (ГПО), для формирования и отработки отказов в моделях приборов и в программных моделях каналов МКИО.*

При разработке моделей необходимо моделировать возникновение ошибок в моделях приборов и в каналах МКИО. Для этого служит специальный класс моделей – класс генераторов потока отказов. Основное отличие ГПО от обычной модели состоит в назначении ГПО: модель ГПО не соответствует ни какому прибору ВС морского НК и предназначен для формирования потока отказов по случайному (вероятностному) или по алгоритмически или таблично заданному закону.

4. *Добавлена поддержка отработки на программных моделях МКИО ошибок в канале, аналогичных ошибкам, отслеживаемым аппаратными адаптерами канала.*

При функционировании ВС морского НК в аппаратных каналах МКИО возможно возникновение различных ошибок. Логика обработки этих ошибок является существенной частью СПО приборов морского НК, управляющих обменом по каналам МКИО. Существует технология внесения ошибок в программную модель[7]. В стенд были встроены средства для внесения ошибок в программную модель канала МКИО. Внесение ошибок выполняется с помощью модели ГПО, присоединенной к специальному интерфейсу программной модели канала МКИО. Внесение ошибок происходит либо в «пакетном» режиме (когда в модели ГПО запрограммирован поток ошибок), либо исследователем в интерактивном режиме во время выполнения эксперимента (когда в модели ГПО отрабатываются изменения

значений управляющих параметров, вносимые исследователем при помощи средства оперативного отображения).

Все ошибки в программной модели канала МКИО рассчитаны либо на "мгновенное" проявление, либо на проявление в течение заданного интервала времени. Мгновенная ошибка должна "накладываться" на моделируемый обмен по каналу МКИО, если на канале происходит обмен в момент прихода соответствующего уведомления об ошибке от ГПО. В этом случае ошибка вносится в этот обмен, иначе никаких действий по внесению ошибки не происходит. Ошибка с заданным интервалом времени будет внесена в ближайший обмен, если он произойдет в течение интервала. На каждый канал может быть запланировано до 50 ошибок различного типа. При превышении этого числа выдается предупреждение. В каждый обмен вносится только одна ошибка.

В программной модели МКИО поддерживаются следующие типы ошибок: обрыв канала, генерация в канале, ошибка четности, неверная пауза перед ответным словом, нарушена непрерывность сообщения, число информационных слов больше заданного, неверный адрес ОУ, неверный тип синхроимпульса, ошибка это контроля при передаче. Эти типы ошибок описаны в[3].

При наличии нескольких запланированных на один интервал времени ошибок, выбор ошибки для наложения на обмен производится по следующему алгоритму:

- выбрать ошибку, которая соответствует более раннему слову в сообщении;
- если несколько ошибок соответствуют одному слову, то ошибки выбираются по порядку, соответствующему порядку, в котором бы такие ошибки обнаружил аппаратный адаптер МКИО[3]

5. *Добавлена поддержка возможности оперативного управления параметрами моделей и формированием ошибок в каналах МКИО в процессе выполнения эксперимента.*

При проведении имитационного эксперимента со стороны исследователя возникает необходимость интерактивно изменять параметры моделей, например для того, чтобы отключать отдельных абонентов на каналах МКИО или управлять внесением неисправностей в обмены по каналам. Поэтому в стенд была добавлена функциональность, позволяющая оперативно менять значения параметров моделей в процессе эксперимента интерактивно.

6. *Реализована возможность запуска эксперимента в режиме «мягкого» реального времени – с ускорением или с замедлением относительно реального масштаба времени.*

При проведении длительных экспериментов, если не предпринимать специальных мер для ускорения эксперимента, процесс моделирования займет столько же времени, сколько и работа реального комплекса. Зачастую нет необходимости запускать модель с соотношением модельного времени к физическому 1:1. В тоже время при запуске модели с ускорением зачастую необходимо оперативно отслеживать изменения параметров. Поэтому в стенд была добавлена возможность запуска эксперимента в ускоренном режиме с оперативным отображением результатов. Помимо ускоренного режима существуют случаи, когда модель необходимо запускать в замедленном режиме, например, если мы хотим отследить изменения многих параметров в течение заданного интервала времени. Для этого была добавлена возможность запуска эксперимента в замедленном режиме. Надо заметить, что обмен по натурному каналу МКИО при замедленном или ускоренном времени выполнения по отношению к реальному невозможен.

7. Добавлена поддержка возможности отключения/включения трассировки событий в динамике выполнения эксперимента.

В процессе работы с моделями фрагментов вычислительных комплексов морских НК выяснилось, что в процессе работы модели могут формироваться трассы эксперимента объемом в несколько ГБайт. Эти трассы имеет высокий уровень детальности информации о событиях на всем протяжении эксперимента, хотя для исследователя детальная трасса на всем интервале времени зачастую не нужна. Поэтому в стенд включён механизм, позволяющий включать или отключать запись событий в трассу для отдельных компонентов в динамике выполнения эксперимента.

Для того, чтобы отключить запись событий в трассу на определённом временном интервале, необходимо в тексте модели в нужный момент отключить запись событий в трассу, а по окончании интервала включить запись событий в трассу.

Заключение

В данной работе представлен стенд полунатурного моделирования, разработанный в Лаборатории вычислительных комплексов факультета ВМК МГУ. Средствами стенда решаются задачи пошагового комплексирования бортовых ВС РВ с применением технологии полунатурного моделирования, а также задачи сравнительной оценки архитектур ВС РВ на ранних этапах разработки.

Стенд применяется в промышленности при разработке бортовых ВС РВ. В данной работе подробно рассматриваются изменения,

внесенные в стенд для его успешного применения при моделировании ВС РВ морских НК.

К перспективным задачам по развитию стенда следует отнести:

- полную реализацию поддержки высокоскоростных каналов Fibre Channel, используемых в новейших ВС РВ;
- применение технологии HLA для реализации взаимодействия со сторонними стендами;
- интеграцию средств оперативного отображения результатов эксперимента с инструментальными средствами отладки в составе ПО приборов ВС РВ, что позволит выполнять оперативное наблюдение за изменением значений переменных в составе ПО приборов ВС РВ.
- реализацию автоматического сравнения результатов регистрации обмена по МКИО с эталонной циклограммой из БД циклограмм;
- создание автономного мобильного варианта стенда для мониторинга и анализа функционирования комплекса на объекте

Литература

1. Грибов Д.И., Смелянский Р.Л. Комплексное моделирование бортового оборудования летательного аппарата // Методы и средства обработки информации. Труды второй Всероссийской научной конференции. - М.: Издательский отдел факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова, 2005. - С.59-74

2. M.P.A.M. Brouwer, et al., Developments in Test and Verification Equipment for Spacecraft. Technical report NLR-TP-2000-658, National Aerospace Laboratory, Netherlands, 2000.

3. ГОСТ Р 52070-2003. Интерфейс магистральный последовательный системы электронных модулей.

4. V.V. Balashov, V.A. Kostenko, R.L. Smeliansky, S.V. Vavinov. A Tool System for Automatic Scheduling of Data Exchange in Real-Time Distributed Embedded Systems. Proc. of the 7th IEEE International Symposium on Computer Networks (ISCN'06), Istanbul, Turkey, 2006.

5. V.V. Balashov, V.A. Kostenko, R.L. Smeliansky, A Tool System for Automatic Scheduling of Data Exchange in Real-Time Distributed Avionics Systems. Proc. of the 2nd EUCASS European Conference for Aerospace Sciences, Brussels, Belgium, 2007.

6. IEEE 1516 Standard Modeling and Simulation (M&S) High Level Architecture (HLA)-Framework and Rules, IEEE, 2000.

7. Волканов Д.Ю. Использование имитационного моделирования для оценки надежности распределенных вычислительных систем // Труды Всероссийской научной конференции "Методы и средства обработки информации" (1 октября - 3 октября 2003 г., г. Москва) -М.: Издательский отдел факультета ВМиК МГУ, 2003. - С. 343-348.

Балашов В.В., Шестов П.Е.

Формирование рекомендаций по обеспечению совместимости требований к обмену по каналу с централизованным управлением во встроенных системах реального времени

Введение

В составе распределенных встроенных систем реального времени (ВСРВ) для взаимодействия между подсистемами широко используются шины (каналы) с централизованным управлением, например мультиплексный канал информационного обмена (МКИО) [1]. Для таких каналов характерно применение статической схемы планирования информационного обмена, при которой расписание обмена составляется заранее и не изменяется в ходе работы ВСРВ. Построение расписания является NP-трудной задачей [2], причем в реальных ВСРВ число работ для включения в расписание достигает нескольких сотен. При построении расписания необходимо учитывать ряд требований к информационному обмену, в том числе специфические для архитектуры конкретной ВСРВ технологические ограничения на обмен по каналу. В связи с высокой вычислительной сложностью задачи построения расписания, невозможно применение точных алгоритмов для решения этой задачи, поэтому на практике используются эвристические алгоритмы планирования [2].

Если используемый алгоритм планирования информационного обмена не может сформировать расписание с учетом всех требований к информационному обмену, будем говорить, что набор требований является несовместимым относительно этого алгоритма. Разработчику ВСРВ необходимо внести изменения в этот набор требований, чтобы обеспечить его совместимость. Актуальна задача автоматического формирования рекомендаций по изменению несовместимого набора требований.

В данной работе предложен алгоритм формирования рекомендаций по изменению несовместимого набора требований к информационному обмену для обеспечения совместимости этого набора. Приведены результаты экспериментального исследования предложенного алгоритма. Предложенный алгоритм применяется в составе инструментальной системы планирования информационного обмена по каналу МКИО [3], используемой в промышленности при проектировании бортовых ВС РВ авиационного и морского назначения.

1. Организация обмена данными по каналу с централизованным управлением

Управление информационным обменом по каналу. Канал с централизованным управлением представляет собой общую шину, к которой подключены абоненты канала. Один из абонентов является контроллером канала, который инициирует все обмены данными между абонентами и осуществляет контроль успешности обменов. Прочие абоненты исполняют команды контроллера, но не могут самостоятельно инициировать обмены данными. Информация по каналу передается в виде сообщений, длительность передачи которых известна при построении расписания. Передачу сообщения по каналу будем называть *работой.*

В статье предполагается, что контроллер функционирует по следующей схеме: (1) в начале работы ВСРВ в контроллер загружается программа, состоящая из цепочек работ; каждая цепочка представляет собой упорядоченный набор работ, которые должны выполняться последовательно; (2) в момент времени между $k * L_{sc}$ и $(k+1) * L_{sc}$ (где L_{sc} – заданный период, измеряемый в тактах работы канала) контроллер начинает выполнение цепочки работ с номером k и доводит его до конца; прерывание этого процесса недопустимо; (3) после завершения цепочки работ и до наступления момента времени $(k+1) * L_{sc}$, контроллер может повторить те работы из цепочки, выполнение которых было неуспешным, например из-за помех в канале. Интервал времени $[k * L_{sc}; (k+1) * L_{sc})$ называется *подциклом* с номером k ($k = 0,1,...$), а значение L_{sc} – *длиной подцикла.* В некоторых подциклах может не присутствовать ни одной цепочки работ.

Требования к информационному обмену по каналу. Типичной рабочей нагрузкой для канала является набор сообщений, предназначенных для периодической передачи; для каждого сообщения задана длительность и требуемая частота передачи. Специфика архитектуры конкретной ВСРВ задает набор технологических ограничений на расписание обмена по каналу. В данной работе будем рассматривать следующие технологические ограничения, применяемые в реальных ВСРВ: (1) максимально допустимое число работ в цепочке (далее – длина цепочки работ); (2) длительность подцикла; (3) доля длительности подцикла, зарезервированная в конце подцикла (например, для повторных обменов); (4) минимально допустимый интервал времени от начала подцикла до начала выполнения цепочки работ (в течение этого интервала контроллер выполняет переключение с предыдущей цепочки на новую, соответствующую начавшемуся подциклу); (5) максимально допустимое отклонение расстояния между

работами по передаче одного и того же сообщения от периода этого сообщения (период – число, обратное частоте передачи сообщения). Формальное представление перечисленных ограничений введено в [4].

Набор сообщений и набор технологических ограничений на расписание в совокупности формируют *набор требований к информационному обмену* по каналу. Каждому требованию в составе такого набора соответствует числовая характеристика – *значение требования*. Под *изменением требования* будем понимать изменение его значения. Изменение требования будем называть *ослаблением*, если любое расписание, удовлетворяющее исходному требованию, удовлетворяет и измененному требованию. Изменение в противоположную сторону будем называть *усилением*. Понятие ослабления и усиление определены для требований, перечисленных в разделе 1 под номерами 1, 3, 4, 5.

2. Задача формирования рекомендаций по обеспечению совместимости требований к информационному обмену

Пусть задан алгоритм планирования информационного обмена A и набор требований к информационному обмену (в том числе значения всех требований). Пусть заданный набор требований несовместим относительно A. В таком случае следует изменить значения некоторых требований из этого набора для обеспечения его совместимости.

При проектировании конкретной ВСРВ одни требования допустимо изменять в рамках некоторых диапазонов (например, максимальную длину цепочки работ), а другие изменять не разрешается (например, длительности и частоты передачи сообщений). Упорядочим требования так, чтобы вначале следовали требования, которые допустимо изменять (пронумеруем их от 1 до NR), а затем все остальные (пронумеруем их от $NR+1$ до $MAXR$). *Набором значений требований* к информационному обмену будем называть вектор, элементами которого являются значения требований, и порядок следования элементов соответствует нумерации требований. Введем функцию $Comp_A(Y)$, определенную на множестве всевозможных наборов значений требований к информационному обмену и принимающую значения: 1 – если набор совместим относительно алгоритма А; 0 – в противном случае.

Введём также следующие обозначения: $init_1,...,init_{MAXR}$ – значения требований из исходного несовместимого набора; $lim_1,...,lim_{NR}$ – значения, до которых допустимо изменять требования с номерами $1..NR$ ($init_i \neq lim_i, i=1..NR$); $INIT$ – вектор с элементами

$(init_1,...,init_{NR})$; LIM – вектор с элементами $(lim_1,...,lim_{NR})$; FIX – вектор с элементами $(init_{NR+1},...,init_{MAXR})$, содержащий значения требований, изменение которых недопустимо; $X=(x_1,...,x_{NR})$ – вектор текущих значений требований, значения которых можно изменять; $X \prec FIX$ – вектор с элементами $(x_1,...,x_{NR},init_{NR+1},...,init_{MAXR})$ (здесь "$\prec$" – операция конкатенации векторов); $INIT^* = (INIT \prec FIX) = (init_1,...,init_{MAXR})$ – вектор значений всех требований из исходного несовместимого набора; $LIM^* = (LIM \prec FIX) = (lim_1,...,lim_{NR},init_{NR+1},...,init_{MAXR})$ – вектор значений требований, в котором все требования, изменение которых допустимо, максимально изменены (ослаблены).

Задачу формирования рекомендаций по обеспечению совместимости требований к информационному обмену будем рассматривать в следующей постановке [5].

Дано: (1) алгоритм A статического планирования информационного обмена; (2) несовместимый относительно A набор значений требований к информационному обмену $INIT^* : Comp_A(INIT^*) = 0$;

(3) интервалы, в которых допустимо варьировать значения требований: $[init_i; lim_i], i = 1..NR$; (4) весовые коэффициенты, отражающие предпочтительность изменения требований: $crel_i \in \mathbf{R}^+$, $i = 1..NR$.

Требуется: найти в допустимых диапазонах такие значения требований, при которых набор требований будет совместимым: $Comp_A(X \prec FIX) = 1$.

Минимизируемый критерий (целевая функция): суммарный «вес» изменений $Cost(X) = \sum_{i=1}^{NR} crel_i * \left|(x_i - init_i)/(lim_i - init_i)\right|$.

Задача может быть представлена в форме *задачи математического программирования* [6] с линейной целевой функцией:

$$\min_{\substack{X \in P \\ Comp_A(X \prec FIX)=1}} Cost(X), \text{ где } P = [init_1; lim_1] \times ... \times [init_{NR}; lim_{NR}].$$

В качестве примера работы, в которой близкая по постановке задача формирования рекомендаций решается для динамического планирования вычислительных задач, можно привести [7].

3. Алгоритм формирования рекомендаций с последовательным сужением области поиска

Структура множества поиска решения. Пусть NR - число изменяемых требований. Множество P, на котором выполняется минимизация функции $Cost$, является многомерным параллелепипедом. Функция $Comp_A$ задана алгоритмически, поэтому неизвестно аналитическое представление множества совместимых наборов требований в составе P, и невозможно разработать алгоритмы формирования рекомендаций, использующие такое представление.

Предлагаемый в данной работе алгоритм разработан исходя из предположения о распределении совместимых наборов на множестве P. Это предположение сформулировано по результатам рассмотрения нескольких десятков двумерных сечений P, построенных для различных наборов сообщений и значений требований к информационному обмену. Все они могут быть представлены в виде объединения следующих подмножеств: (1) односвязное множество совместимых решений, в которое, в частности, входит набор LIM (обозначим это множество за $COMP1$); (2) односвязное множество несовместимых решений, в которое входит набор $INIT$ (обозначим это множество за $INCOMP1$); (3) множество решений из P, не входящих ни в $COMP1$, ни в $INCOMP1$, и представляющее собой комбинацию совместимых и несовместимых решений. При этом множество $COMP1$ содержит значительную часть от общего числа совместимых наборов (не менее 30-40%).

При разработке алгоритма решения задачи формирования рекомендаций будем исходить из предположения о том, что множество P для 3 и более требований допускает аналогичное деление на подмножества.

Схема работы алгоритма формирования рекомендаций. Предлагаемый алгоритм формирования рекомендаций выполняет сужение множества поиска решения с учётом линейности целевой функции $Cost$. При нахождении очередного совместимого решения X, алгоритм продолжает поиск решений на множестве наборов требований, имеющих значение, меньшее, чем $Cost(X)$. Далее будем использовать обозначение $P(C)$ для множества наборов требований из P, значение $Cost$ на которых не превышает константы C ($C \geq 0$).

В данной работе предлагается следующая схема работы алгоритма формирования рекомендаций:

1. Первичное сужение области поиска, которое выполняется в предположении, что доля совместимых решений в множестве P достаточно велика: (i) сужение на основе случайного выбора решений

(процедура *rnd_narrowing*); (*ii*) уточнение результатов процедуры *rnd_narrowing* посредством сдвига по направлению наискорейшего убывания функции *Cost* (процедура *guided_correction*).

2. Поиск решения в суженной области на основе усиления отдельных требований (циклическое выполнение процедуры *percoord_strengthening*).

Завершение цикла на этапе 2 происходит, если в течение очередной итерации значение *Cost* на лучшем (минимальным) по *Cost* найденном совместимом решении уменьшилось менее чем на *delta* ($delta > 0$ – параметр алгоритма).

Алгоритм является завершимым, поскольку: завершимы процедуры *rnd_narrowing*, *guided_correction*, *percoord_strengthening* (обоснование завершимости см. описание этих процедур далее в текущем разделе); число итераций цикла на этапе 2 конечно (не превышает $\lceil Cost(LIM)/delta \rceil$).

Далее будем считать, что для требований, принимающих вещественные значения, введена равномерная целочисленная сетка.

Процедура *rnd_narrowing* **сужения множества поиска на основе случайного выбора решений.** Входными данными для процедуры являются: *NMISS* – ограничение сверху на число полученных подряд несовместимых решений; *NKEEP* – ограничение сверху на число запоминаемых лучших совместимых решений; *TOPCOST* – начальное ограничение сверху на значение функции *Cost*, задающее начальное сужение множества поиска (при вызове процедуры *rnd_narrowing* на этапе первичного сужения области поиска, $TOPCOST = Cost(LIM)$).

Набор выходных данных процедуры *rnd_narrowing* следующий: *BESTSET* – множество лучших совместимых решений, запомненных в ходе выполнения процедуры; *OUTCOST* – ограничение сверху на значение функции *Cost*, полученное по результатам выполнения процедуры и определяющее сужение множества поиска.

Процедура *rnd_narrowing* завершается при выполнении хотя бы одного из следующих условий: (1) в цикле случайного выбора решений получены подряд *NMISS* несовместимых решений; (2) перебраны все решения, принадлежащие текущему множеству поиска.

Схема выполнения процедуры *rnd_narrowing* следующая:

i) выполнить инициализацию: множество лучших совместимых решений: $BESTSET := \varnothing$; множество перебранных решений: $ENUM := \varnothing$; текущее ограничение на значение *Cost*: $C := TOPCOST$; счётчик несовместимых решений: $CNT := 0$;

ii) если множество $P(C) \setminus ENUM$ пусто, перейти к шагу *viii*;

iii) случайным образом выбрать решение X из множества $P(C) \setminus ENUM$;

iv) $ENUM := ENUM \cup \{X\}$;

v) если решение X совместимо, то:

- установить новое ограничение на множество поиска: $C := Cost(X)$;
- если $|BESTSET| < NKEEP$, то $BESTSET := BESTSET \cup \{X\}$;
- иначе положить Y равным элементу $BESTSET$ с максимальным по $BESTSET$ значением $Cost$; если $Cost(X) < Cost(Y)$, то $BESTSET := (BESTSET \setminus \{Y\}) \cup \{X\}$;

vi) если решение X несовместимо, то:

- увеличить счётчик несовместимых решений: $CNT := CNT + 1$;
- если $CNT > NMISS$, перейти к шагу *viii*;

vii) перейти к шагу *ii*;

viii) $OUTCOST := C$; завершить работу.

Процедура *rnd_narrowing* является завершимой, поскольку число элементов в множестве P конечно (с учётом целочисленного представления значений всех требований), и ни один из них не перебирается в процедуре более одного раза.

Процедура *guided_correction* уточнения совместимых решений сдвигом по направлению наискорейшего убывания целевой функции. При выполнении данной процедуры, решения задачи формирования рекомендаций рассматриваются как элементы пространства $\mathbf{R}^{NR}$ (далее – *точки*). При определении совместимости точки выполняется округление каждой её координаты до ближайшего целого значения, затем определяется совместимость полученного набора требований. Линейная функция *Cost* естественным образом расширяется на $\mathbf{R}^{NR}$, и в каждой точке этого пространства имеет однозначно определённое направление наискорейшего убывания, не зависящее от выбранной точки. Обозначим за D нормированный вектор, задающий это направление. Обозначим также $G(C) = \{X \in \mathbf{R}^{NR} \mid Cost(X) = C\}$.

Входными данными для процедуры являются: X^{est} – начальное приближение (совместимый набор требований); C^* – положительная константа, ограничивающая сверху значение функции *Cost* на наборе-результате процедуры; d – приращение для значения функции *Cost* ($d > 0$).

На выходе процедуры *guided_correction* – совместимый набор требований X^{res}, значение *Cost* на котором не превышает C^*.

Процедура *guided_correction* завершается в случае, если ни от одной из найденных совместимых точек не удалось продолжить сдвиг вдоль вектора D без потери совместимости.

Формальное описание схемы выполнения процедуры *guided_correction* требует применения инструментария линейной алгебры и введения большого количества обозначений, поэтому ниже приведены только основные шаги процедуры:

i) инициализировать текущее приближение: $X := X^{cn}$; если $Cost(X) > C^*$, выполнить сдвиг X вдоль D так, чтобы $Cost(X)$ стало равно C^*; в случае выхода X за границы P в результате этого сдвига, **завершить процедуру неуспешно**;

ii) выполнять сдвиг X вдоль D (на каждом шаге сдвига приращение *Cost* должно быть равно d), пока не будет достигнута несовместимая точка или граница P;

iii) выполнить поиск совместимых точек на $G(C)$, где C – значение *Cost* на последней полученной в процессе сдвига вдоль D совместимой точке (обозначим её за X^{curr});

iv) если множество *CAND* непусто, то: выполнить для каждого элемента *CAND* сдвиг вдоль D аналогично шагу *ii*; среди полученных точек выбрать в качестве текущего приближения точку с наименьшим значением *Cost*; перейти к шагу *iii*;

v) иначе:

- если $Cost(X^{curr}) < C$, то: положить X^{res} равным результату покоординатного округления X^{curr}; **завершить работу успешно**;
- иначе **завершить работу неуспешно**.

Процедура *guided_correction* является завершимой, поскольку: значение $Cost(LIM)$ конечно (следовательно, конечно число шагов сдвига текущего приближения вдоль вектора D при каждом выполнении шага *ii* или *iv*; это число ограничено сверху значением $\lceil Cost(LIM)/d \rceil$); число выполнений шага *iii* не превышает $\lceil Cost(LIM)/d \rceil$; число элементов множества *CAND* конечно и не превышает числа изменяемых требований; процесс построения множества *CAND* завершим (детальное описание процесса поиска совместимых решений на $G(C)$ выходит за рамки данной статьи).

Процедура *percoord_strengthening* поиска совместимых решений на основе усиления отдельных требований. Данная процедура выполняет поиск совместимых решений на подмножествах P, на которых отдельные требования имеют максимально слабые

значения (для каждого подмножества такое требование единственно). Поиск выполняется с минимальным (единичным) шагом, поскольку пересечение изолированной совместимой области с одним из упомянутых подмножеств P может содержать всего несколько решений. Затем процедура последовательно усиливает требования в найденных решениях для снижения значения *Cost* на них.

На входе процедуры *percoord_strengthening* – начальное приближение X^{cur} (совместимый набор требований). На выходе – совместимый набор требований X^{res}. Процедура завершается по завершении перебора усиливаемых требований.

Схема выполнения процедуры следующая:

i) инициализировать множество решений-кандидатов: $CAND := \varnothing$;

ii) для k от 1 до NR:

   a) инициализировать текущее приближение: $X := X^{cur}$;

   b) установить k-е требование в максимально слабое значение: $x_k := x_k^{est}$;

   c) зафиксировать последовательность усиления требований: $(n_1,...,n_{NR}) : n_1 \neq k$;

   d) инициализировать счётчик требований: $i := 1$;

   e) усиливая с единичным шагом требование с номером n_i в составе X, найти наиболее сильное значение этого требования, при котором X является совместимым; если такое значение не найдено, продолжить цикл по k;

   f) повторять шаг e), перебирая требования в последовательности, зафиксированной на шаге c);

   g) полученный совместимый набор включить в множество $CAND$;

iii) если множество $CAND$ пусто, **завершить работу неуспешно**;

iv) положить X^{res} равным набору из $CAND$, имеющему минимальное по $CAND$ значение *Cost*; **завершить работу успешно.**

Процедура *percoord_strengthening* является завершимой, поскольку: число возможных последовательностей перебора требований конечно; диапазон изменения для каждого требования конечен.

4. Экспериментальное исследование алгоритма формирования рекомендаций

Задачи исследования. В рамках данной работы было проведено экспериментальное исследование предложенного алгоритма формирования рекомендаций. В рамках исследования решались следующие задачи: (1) оценить точность алгоритма (отклонение

значения *Cost* на найденном решении от оптимального значения); (2) оценить стабильность работы алгоритма (стабильность определяется разбросом значений *Cost*, получаемых при последовательных прогонах алгоритма на одних и тех же исходных данных); (3) исследовать зависимость стабильности работы алгоритма от размерности задачи (числа изменяемых требований); (4) сравнить алгоритм по точности и стабильности с более простыми алгоритмами, основанными на случайном выборе решений с сужением множества поиска, а именно:

i) алгоритм со случайным выбором решения и сужением области поиска (аналогичен процедуре *rnd_narrowing*);

ii) алгоритм, аналогичный указанному в пункте *i*, но выполняющий дополнительное уточнение каждого найденного совместимого решения сдвигом по направлению наискорейшего убывания целевой функции (аналогично шагам *i*, *ii* процедуре *guided_correction*) и сужающий множество поиска по результатам уточнения каждого решения.

При сравнении предлагаемого алгоритма по точности с более простыми алгоритмами, последние выполнялись на тех же исходных данных, с ограничением: число перебираемых наборов требований должно совпадать с числом наборов, перебранных предлагаемым алгоритмом при работе на этих исходных данных. Данное ограничение позволяет провести сравнение характеристик алгоритмов при условии практически равных вычислительных затрат на выполнение каждого алгоритма.

Исходные данные для экспериментов. Для исследования были подготовлены наборы сообщений, по своим характеристикам (число сообщений, их длительности и частоты) аналогичные реальным наборам сообщений для передачи по каналу с централизованным управлением (канал МКИО) в составе ВС РВ. Для построения расписания при оценке совместимости наборов требований (вычисления функции $Comp_A$) применялся алгоритм, реализованный в инструментальной системе «САПР циклограмм» [3], автоматизирующей планирование информационного обмена по каналу МКИО.

Результаты экспериментов. Для задачи размерности 3, на отдельных наборах исходных данных методом полного перебора были найдены оптимальные решения. Для этих же наборов исходных данных были последовательно выполнены 50 прогонов каждого из 3 сравниваемых алгоритмов. С применением метода проверки статистических гипотез [8] для каждой пары {алгоритм; набор исходных данных} был определён доверительный интервал, в который с вероятностью 0.95 попадают значения функции *Cost* на 90% получаемых алгоритмом результатов. Затем были рассчитаны значения относительного отклонения верхних границ доверительных интервалов от оптимального

значения $Cost$: $Dev = (X^* - X_{opt})/X_{opt}$, где X^* – верхняя граница доверительного интервала, а X_{opt} – оптимальное значение $Cost$.

На рис. 1 показаны значения отклонения, полученные на различных наборах исходных данных. На наборах 1, 3, 5 предложенный алгоритм находит оптимальное решение. На наборе 4 отклонение для предложенного алгоритма составило 25%, но это значительно меньше, чем у двух других алгоритмов на том же наборе.

Рис. 2. Относительное отклонение верхней границы доверительного интервала от оптимального решения

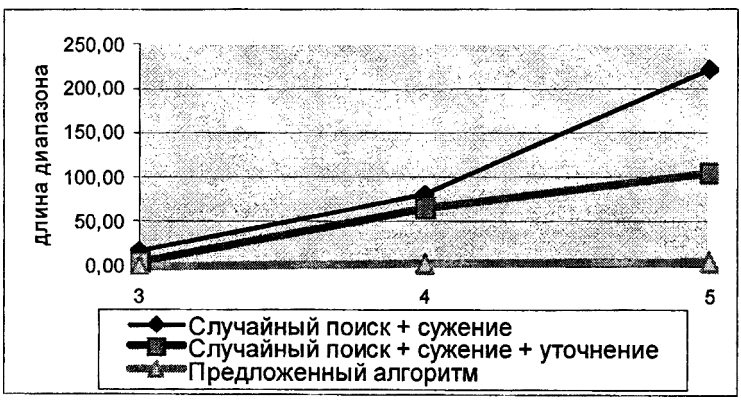


Рис. 3. Зависимость длины доверительного интервала от размерности задачи

Для набора 5 были также проведены эксперименты на задачах размерности 4 и 5. В этих случаях получить точное решение перебором практически невозможно, поэтому предметом исследования являлся разброс значений целевой функции на получаемых решениях. Разброс определялся как длина доверительного интервала, определяемого по методу проверки статистических гипотез. На графике (рис. 2) показана зависимость длины интервала для целевой функции при увеличении размерности задачи с 3 до 5. Видно, что для предложенного алгоритма длина интервала остаётся очень небольшой (0, 3, 5 единиц), в то время как для двух других алгоритмов быстро возрастает до практически неприемлемых значений.

5. Заключение

В данной работе предложен алгоритм формирования рекомендаций по обеспечению совместимости требований к обмену по каналу с централизованным управлением в ВС РВ. Алгоритм основан на сочетании схем случайного поиска с сужением множества поиска и направленного изменения текущего решения. Приведены результаты экспериментального исследования алгоритма на данных, аналогичных по характеристикам рабочей нагрузке на канал в реальной ВС РВ. Проведённые эксперименты показали высокую точность и стабильность работы алгоритма. Дальнейшее развитие алгоритма запланировано в рамках работ по подсистеме формирования рекомендаций инструментальной системы «САПР циклограмм» [3], применяемой при проектировании бортовых ВС РВ для авиационных и морских бортов.

Литература

1. Государственный стандарт РФ «Интерфейс магистральный последовательный системы электронных модулей» ГОСТ Р 52070-2003.

2. Костенко В.А., Гурьянов Е.С. Алгоритм построения расписаний обменов по шине с централизованным управлением и исследование его эффективности // Программирование, 2005, № 6.

3. Balashov V.V., Kostenko V.A., Smeliansky R.L. et al. A Tool System for Automatic Scheduling of Data Exchange in Real-Time Distributed Embedded Systems // Proc. Seventh IEEE Internat. Sympos. on Computer Networks (ISCN'06). Istanbul, Turkey, 2006.

4. Балашов В.В. Формирование рекомендаций по изменению требований к информационному обмену при невозможности построения циклограммы обменов по мультиплексному каналу // Тр. Третьей междунар. конф. "Параллельные вычисления и задачи управления" (PACO'2006). М.: ИПУ РАН, 2006. С. 422-437.

5. Балашов В.В. Алгоритмы формирования рекомендаций при планировании информационного обмена по каналу с централизованным управлением. // Известия РАН. Теория и системы управления, 2007, N.6, с. 76-84.

6. Мину М. Математическое программирование. Теория и алгоритмы. М.: Наука, 1990.

7. Stewart D.B., Arora G. A Tool for Analyzing and Fine Tuning the Real-Time Properties of an Embedded System // IEEE Trans. on Software Engineering. Issue 4. 2003. V. 29.

8. Калинина В.Н., Панкин В.Ф. Математическая статистика // М.: Высшая школа, 1998.

Гамаюнов Д.Ю., Сапожников А.В.

Обнаружение аномального поведения приложений на уровне ядра операционной системы

Введение

Большинство современных систем защиты информации в сетях – антивирусы, системы обнаружения и предупреждения атак и т.п. - используют один из двух подходов: обнаружение злоупотреблений или обнаружение аномалий.

Первый подход чаще всего применяется в системах обнаружения атак (например, популярный Snort [1]) и в антивирусах. В основе этого подхода [2], [3] лежит использование заранее заданных описаний вредоносных воздействий на компьютерную систему: сетевые атаки на конкретные уязвимости приложений, «тела» компьютерных вирусов, червей и т.п. Описание одной атаки или вируса, как правило, называют «сигнатурой».

Один из главных недостатков подхода обнаружения злоупотреблений заключается в том, что он принципиально не позволяет обнаруживать неизвестные атаки или вирусы. Использующие данный подход системы обычно работают в "догоняющем режиме" и вынуждены постоянно обновлять свои базы сигнатур. Составлением новых сигнатур, позволяющих обнаруживать новые версии вредоносного программного обеспечения (ПО) или новые сетевые атаки занимаются квалифицированные эксперты.

Альтернативным обнаружению злоупотреблений является подход обнаружения аномалий [2], [3]. Идея этого подхода проста: нам не известно множество возможных злоупотреблений против защищаемой системы, но известно, как защищаемая система должна работать в штатном режиме. Тем самым в функционировании системы или ее компонентов можно выделить два типа поведения: нормальное поведение и аномальное (т.е. отклоняющееся от нормального). Задача систем, основанных на обнаружении аномалий – это наблюдение за поведением защищаемой системы с целью выявления отклонений наблюдаемого поведения от нормального.

Аномалии в функционировании программ или системных компонентов могут иметь различную природу: они могут быть вызваны системными сбоями, ошибками в программе, либо являться результатами целенаправленного вредоносного воздействия – например, последствиями атак на приложение со стороны злоумышленника. Даже не имея возможности определить, что именно вызвало наблюдаемое

отклонение от нормального поведения, системы обнаружения аномалий могут принять соответствующие меры: например, уведомить системного администратора о нештатной ситуации или автоматически заблокировать выполнение программой нежелательных действий.

Для того чтобы использовать системы, основанные на обнаружении аномалий, эксперт должен составить так называемый *профиль* нормального поведения для защищаемой системы или отдельного приложения, являющегося объектом защиты. При этом профиль может быть составлен как вручную, на основании информации о штатном поведении программы (например, если речь идет о сетевых сервисах, реализующих известные сетевые протоколы), либо с помощью автоматизированных профилировщиков. В последнем случае информация о том, что же является нормальным поведением программы, собирается на основании наблюдения за поведением программы в штатном режиме.

В настоящее время подход к описанию нормального поведения защищаемой системы и обнаружению отклонений от нормального поведения реализован в ряде академических разработок ([3],[4],[5]) и некоторых системах, применяемых на практике. Из практически применяемых систем наиболее известны (и соответственно получили наиболее широкое распространение) две: SELinux и AppArmor.

В данной статье описывается предложенная нами модификация подхода, используемого в AppArmor, в которой единичный профиль AppArmor привязывается не к программе целиком, а к отдельным состояниям программы, а обнаружение аномального поведения приложений осуществляется на основе наблюдения за изменением состояния программы. Предлагаемая модификация основана на формальной модели описания поведения сетевых объектов, разработанной в работе [6]. Основная цель предлагаемой модификации - повышение точности описания нормального поведения сетевых сервисов в AppArmor, однако может применяться и для других приложений, для которых существует спецификация поведения. В дальнейшем планируется реализовать поддержку модифицированного подхода AppArmor в систему защиты SELinux.

1. Подходы, используемые в SELinux и AppArmor

Система SELinux [8], [9], [10], разработанная Агентством Национальной Безопасности США, входит в официальное ядро Linux, начиная с версии 2.6. Технически SELinux реализован через интерфейс модулей безопасности (LSM, Linux Security Modules). Функционирование механизма ограничения доступа SELinux выглядит следующим образом: всякий раз, когда процесс обращается к

некоторому ресурсу, интегрированный в ядро менеджер ресурсов отправляет запрос серверу безопасности, хранящему так называемую *политику безопасности* для защищаемой системы. Политика безопасности явно сопоставляет объектам и субъектам системы типы и домены, а так же определяет правила доступа субъектов с заданными доменами к объектам заданных типов. На основании этих правил сервером политики безопасности принимается решение о предоставлении доступа или отказе, которое исполняется сервером ограничения доступа (через который и осуществляются все запросы в системе). Рисунок 1 иллюстрирует данную схему работы SELinux.

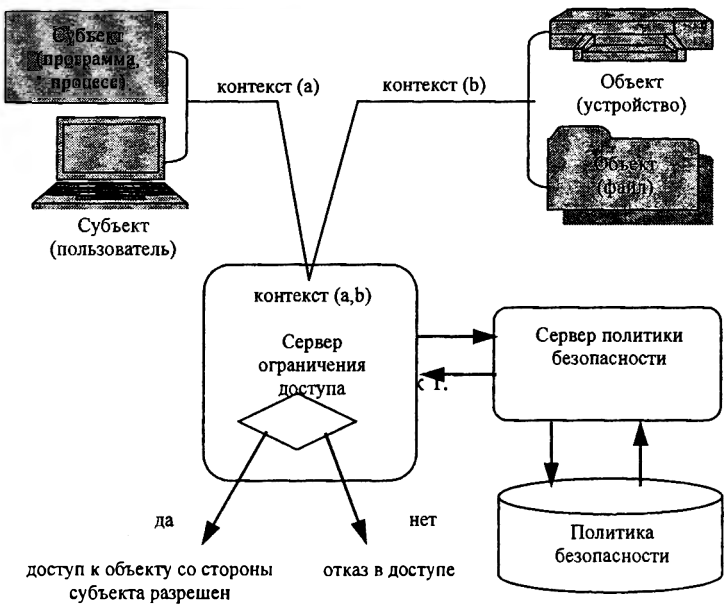


Рисунок 1. Пример обработки запроса к ресурсу в SELinux

С содержательной точки зрения SELinux реализует мандатный контроль доступа (MAC, Mandatory Access Control) поверх стандартного дискреционного механизма доступа. Мандатный контроль доступа реализуется в SELinux через три основных механизма: принудительный механизм типизации (TE, type enforcment), ролевое управление доступом (RBAC, role-based access control), и многоуровневую систему безопасности (MLS, multi-level security). Основой модели безопасности SELinux являются *контексты безопасности* – наборы атрибутов, ассоциированных с каждым субъектом (процессами) и объектом (файлами, сокетами и т.п.). Для каждого объекта системы и каждого

приложения определяются тип, домен и роль данного объекта или приложения. При этом тип и домен, по сути, являются разными названиями одной и той же сущности: домены ассоциируются с субъектами, типы – с объектами защищаемой системы.

Формирование списков пользователей, ролей и типов производится с помощью их явного описания в специальных конфигурационных файлах. Эти файлы включают в себя:

- Список пользователей, в котором каждой пользовательской записи сопоставляется множество ролей, в которых могут выступать ассоциированные с пользователем субъекты (т.е. программы),
- Список ролей, для каждой из которых указывается список доменов, в которых могут выполняться приложения с данной ролью, либо список типов, к которым могут принадлежать объекты системы с данной ролью,
- Список типов, с указанием того, какие объекты могут быть ассоциированы с данным типом (например, файлы, порты и т.п.).

Каждому объекту и субъекту системы (например, файлам с данными, программам и т.п.) в специальном конфигурационном файле присваиваются их контексты безопасности, включающие в себя идентификаторы пользователя, роль и тип (или домен).

Примеры описания пользователей, ролей и типов, а так же присвоения объектам системы контекстов безопасности приведены ниже.

```
user root roles { user_r sysadm_r system_r};
# данная запись вводит пользователя системы root,
который может выступать в одной из перечисленных
ролей (обычного пользователя, администратора
системы или в особой роли с названием system_r)

role system_r types { httpconfig_t};
# данная запись указывает, что пользователь с ролью
system_r имеет доступ к домену httpconfig_t.

type httpconfig_t, file_type;
# вводится тип httpconfig_t, который может быть
приписан объектам-файлам.

type http_port_t, port_type;
# вводится тип http_port_t, который может быть
приписан объектам, являющимся сетевыми портами.

/usr/http/conf(/.*)  --
system_u:system_r:httpconfig_t
# данная запись указывает, что все файлы,
удовлетворяющие соответствующему регулярному
```

выражению, имеют контекст безопасности с пользователем system_u, и описанными ранее ролью system_r и типом httpconfig_t.

На этапе формирования политики безопасности SELinux задаются правила, регламентирующие доступ субъектов системы, принадлежащих определенным доменам, к объектам, имеющим определенные типы. Все разрешенные действия должны быть явно заданны в виде правил. Действия, для которых нет явных правил – запрещаются. Пример правила приведен ниже:

allow httpd_t httpconfig_t:file { read getattr }

Данное правило позволяет приложениям, находящимся в домене httpd_t, читать объекты-файлы, имеющие тип httpconfig_t и узнавать атрибуты таких файлов. Если соответствующему http-демону будет разрешено находится в домене httpd_t, то он сможет в нашем примере читать файлы конфигурации из соответствующей директории.

Кроме разрешения на доступ субъектов к объектам правила SELinux так же явно указывают разрешение на присвоение вновь запускаемым процессам определенных доменов.

Конечная цель настройки SELinux на защищаемой системе заключается в определении множеств пользователей, ролей и типов (доменов), создании множества правил, разрешающих все необходимые для нормального функционирования системы виды доступа субъектов к объектам.

Описанную модель мандатного доступа расширяет *многоуровневая система безопасности*, которая добавляет к контексту безопасности объектов и субъектов защищаемой системы так называемые *уровни* и *ранги*. Этот механизм предназначен для защиты конфиденциальности информации: так, объект файловой системы может быть прочитан только субъектами с равным или большим уровнем доступа; а уровень доступа к объектам, созданным любым процессом, не может быть ниже, чем у его создателя – для предотвращения его чтения субъектами с меньшим уровнем допуска, чем у создателя.

Процесс настройки SELinux для конкретной защищаемой системы (т.е. создание политики безопасности, специфицирующей допустимые операции доступа) до недавнего времени был крайне трудоемким. Модель «запрещено все, что не разрешено явно» требовала создания правил, разрешающих все допустимые операции доступа в защищаемой системе. Сравнительно недавно в пакет SELinux стали включаться инструменты, частично автоматизирующие процесс настройки системы, а так же были добавлены упрощенные политики

безопасности. Упрощенные политики безопасности позволяют наложить ограничения на основной, общий для большинства систем набор программ (включая основные сетевые сервисы), присвоив остальным объектам и субъектам специальный тип «unconfined_t». Объекты и субъекты данного типа не подвергаются ограничениям (кроме стандартных ограничений дискреционной модели доступа Linux).

AppArmor [11], [12], приобретенный компанией Novell, включен в настоящее время в OpenSUSE, а его ранние версии доступны в открытом виде. Используя для обеспечения безопасности ту же идею, что и SELinux – предоставление для каждого приложения минимального набора привилегий, необходимых для корректной работы – AppArmor реализует существенно более простой подход для контроля доступа. AppArmor защищает не всю систему в целом, а отдельные приложения и сервисы, явно специфицируя нормальное поведение каждого из них в терминах разрешенных операций доступа.

Для описания нормального поведения программ в AppArmor используются так называемые *профили,* каждый из которых явно указывает для защищаемого приложения набор действий, которые считаются нормальными и разрешены к выполнению. Все, что не разрешено явно – запрещается моделью безопасности AppArmor.

В настоящее время профили AppArmor, могут содержать правила, регламентирующие следующие аспекты поведения приложений:

- правила определяют, к каким объектам файловой системы и с какими правами (на чтение, запись и т.п.) приложение имеет доступ. Такие объекты задаются посредством указания путей к ним, при этом допускается использование регулярных выражений;
- правила специфицируют, какие POSIX.1e capabilities [7] приложение может иметь. Примерами таких capabilities могут быть CAP_SETUID, CAP_NET_RAW и т.п;
- в случае, если приложение имеет право запускать некоторые файлы на исполнение, существуют правила, которые определяют, какие профили будут использоваться для защиты дочерних процессов. Дочерние процессы могут использовать профиль родительского процесса, свой собственный профиль AppArmor или запускаться в неконтролируемом режиме, без защиты профилем безопасности;
- кроме того, в последних версиях AppArmor профили могут также содержать правила для сетевых операций, которые может выполнять приложение. Правила, определяющие доступные сетевые операции, похожи на стандартные упрощенные правила межсетевых экранов.

Модель функционирования AppArmor в защищающем режиме выглядит следующим образом: для каждого ограниченного профилем приложения AppArmor отслеживает выполняемые им действия. Каждый запрос к системе сопоставляется с профилем. Если соответствующая запись входит в профиль – действие разрешается. В противном случае приложение получает отказ в доступе.

Профили приложений создаются автоматизированным образом в режиме обучения на основе наблюдения за функционированием приложения. Специальное инструментальное средство создания профилей в режиме обучения сохраняет записи о тех действиях, которые выполняет приложение в режиме штатного функционирования. Системный администратор далее имеет возможность включить каждое из наблюдавшихся действий в профиль нормального поведения для данного приложения, запретить выполнение данного действия (не включив его в профиль), модифицировать действие (например, обобщив или уточнив список доступных приложению файлов с помощью модификации регулярного выражения и т.п.). Также имеется возможность редактирования профилей, позволяющая добавлять новые правила или изменять существующие. Пример профиля AppArmor для веб-браузера Firefix приведен ниже:

```
/usr/lib/firefox/firefox.sh                    {
...
/bin/basename                                mixr,
/usr/lib/firefox/*                             r,
/usr/lib/firefox/firefox-bin                  px,
...
}
```

Данный профиль предназначен для защиты приложения firefox.sh, позволяя ему читать любые файлы в директории usr/lib/firefox, а так же разрешая запускать приложение firefox-bin (которое при запуске будет защищено своим собственным профилем). Этот профиль так же разрешает приложению firefox.sh читать и запускать приложение basename, которое при запуске «унаследует» данный профиль, т.е. будет подвергнуто таким же ограничениям.

Так как AppArmor использует модель «запрещено все, что не разрешено явно», то данный профиль позволяет существенно ограничить приложение firefox.sh, в частности не позволяя ему читать какие-либо файлы, кроме файлов в заданной директории, а так же модифицировать любые файлы или запускать программы, кроме двух явно заданных.

2. Предлагаемая модификация подхода, используемого в AppArmor

Достоинствами подхода, используемого в AppArmor, являются простота описания нормального поведения в виде индивидуальных профилей для каждого приложения и возможность автоматизации процесса построения профилей. В частности, для создания профиля приложения, нам не требуется знание о внутренней структуре приложения; не требуется также внесение каких либо изменений в защищаемое приложение и как следствие наличие его исходных кодов.

Однако данный подход имеет существенный недостаток. Защищаемое приложение с точки зрения AppArmor представляет собой "черный ящик". AppArmor не способен различить внутренние состояния приложения. Тем самым, при составлении профиля нормального поведения, каждое из наблюдаемых действий приложения ассоциируется именно с приложением, и запрещается или разрешается для всего приложения сразу, все зависимости от того, на каком этапе выполнения находится приложение, каково его внутреннее состояние и каково состояние окружения.

Тем не менее, для приложений и сетевых сервисов с нетривиальной логикой работы, типичны случаи, когда можно выделить множество состояний, существенно различающихся операциями доступа, которые в этих состояниях считаются нормальными и могут быть разрешены. Иными словами, профили AppArmor могут разрешать приложению иметь больше прав доступа и привилегий, чем реально необходимо для нормальной работы.

Эта проблема осознанна и частично решается самими разработчиками AppArmor. В последних версиях AppArmor для веб-сервера Apache добавлена возможность применения к отдельным частям кода сетевого приложения особой части правил, выделенных из основного профиля в подпрофиль. В профилях AppArmor допускается создание подпрофилей (но без дальнейшей вложенности), содержащих произвольные наборы правил. Подобные подпрофили применяются в той ситуации, когда веб-сервер выполняет код скриптов, предназначенных для обработки запросов к определенным URI. Для такого кода оказывается полезным определить свои наборы разрешенных операций, отличающиеся от базового набора доступных веб-серверу привилегий.

Подпрофили адресуются в рамках базового для приложения профиля с помощью указания URI, обработкой которого должен заниматься код, подлежащий ограничению с помощью подпрофиля. Для того чтобы AppArmor смог выбрать правила из подпрофиля вместо основного профиля, само защищаемое приложение должно уведомить

AppArmor о начале и конце выполнения выделенного особого кода. В текущих версиях AppArmor это реализуется через особый модуль к Apache mod_change_hat, через который приложение может вызвать специальную функцию change_hat(), изменяющую применяемый профиль на подпрофиль и обратно. Пока данная возможность – наличие подпрофилей для описания штатного выполнения отдельных участков кода приложения – реализована только для Apache. Использование механизма подпрофилей требует модификации исходной программы, являющейся объектом защиты (для Apache реализовано с помощью модуля mod_change_hat).

Нами предлагается модификация самого подхода, используемого AppArmor, которая является универсальной (в смысле применимости к различным программам) и не требует модификации исходных программ. Предлагаемая модификация применима (т.е. дает преимущество в смысле точности профилировки перед не модифицированным подходом) для случаев, когда внутренняя структура и логика работы приложения известна. Типичным примером таких приложений с известной логикой работы являются сетевые сервисы, реализующие известные сетевые протоколы. Идея предлагаемой нами модификации состоит в том, чтобы разбить единое общее множество правил, разрешающих те или иные действия приложения, на множества правил, каждое из которых будет соответствовать одному из состояний защищаемого приложения или одному из переходов между двумя состояниями.

Предлагаемый нами метод основан на формальной модели функционирования сетевых объектов, разработанной в рамках [6]. В данной формальной модели состояние информационной системы описывается, как множество входящих в систему объектов и их состояний:

$\theta = < \Re, \{S_r \mid r \in \Re\} >$, где $\Re$ – множество объектов системы, а S_r - состояние для соответствующего объекта r.

Под состоянием объекта понимается его показатель загруженности и дерево доступа к другим объектам, т.е. состояние S_r для объекта r определяется как:

$S_r = (In(r), Out(r), I(\xi(r)))$, где $In(r)$ - множество объектов, осуществляющих доступ к r, $Out(r)$ - множество объектов, к которым осуществляет доступ r, а $I(\xi(r))$ - индикатор загруженности объекта, принимающий одно из предопределенных значений.

Переходы между состояниями объекта происходят в результате осуществления операций доступа к данному объекту или со стороны данного объекта. Поведением объекта в этой модели называется

множество траекторий объекта – конечных непустых цепочек его состояний (или, что эквивалентно, переходов между ними и заданных начальных состояний).

$t_r = S_1, S_2, ..., S_k$ – трасса объекта (непустая, замкнутая слева последовательность состояний объекта). Трасса объекта, заданная в виде последовательности состояний, однозначно описывает все операции доступа к объекту со стороны других объектов и операции доступа самого объекта к другим объектам системы (и обратно, последовательность операций доступа однозначно определяет трассу как последовательность состояний). Поведение объекта определено как множество всех его возможных траекторий: $Bh(r) = \{t_r\}$.

Задача обнаружения аномального поведения (т.е. отклонения от нормального поведения) решена в рамках данной модели путем сведения ее к задаче распознавания цепочек символов, не принимаемых конечной машиной состояний, принимающей заданное множество трасс, соответствующих нормальному поведению системы.

Формально задача обнаружения аномалий формулируется следующим образом:

- Заданно описание нормального поведения B,

  $B = \{Bh(r) \mid r \in \Re : \forall S \in t_r, S \in N_r\}$, где N_r- множество нормальных (допустимых) состояний для ресурса r. Иными словами нормальное поведение заданно в виде множества поведений объектов, состоящих из трасс, включающих только допустимые состояния для соответствующих ресурсов.

- Заданна последовательность состояний наблюдаемой системы $T_{sys} = \theta_1, \theta_2, ..., \theta_N$.

Требуется найти множество объектов $O^* = \{o\} \subset \Re$, и соответствующее ему множество траекторий $T^* = \{t_o \mid o \in O^*\}$, которое не принадлежит описанию нормального поведения.

В работе [6] показано, как может быть решена приведенная выше задача с помощью конечных автоматов специального вида, которые могут быть построены посредством автоматизированной процедуры.

Принципиальным моментом в данной формальной модели является то, что под состоянием объекта (т.е. программы) понимается не состояние ее графа выполнения и состояние памяти, а наборы объектов, к которым или со стороны которых осуществляются операции доступа. Поведение объекта защищаемой системы так же описывается в терминах операций доступа, осуществляемых данным объектом или к данному объекту. Механизм обнаружения аномалий в рамках данной формальной модели основан на механизме конечных автоматов,

распознающих цепочки символов, алфавитом для которых выступает множество возможных операций доступа. Отметим отдельно, что подходы, применяемые в SELinux и AppArmor, так же описывают нормальное поведение программ в терминах операций доступа к объектам.

Нами предлагается модификация существующего подхода AppArmor, в которой с помощью механизма конечных автоматов реализуется описание и распознавание допустимого поведения приложения как множества состояний и переходов между ними. Иными словами, в функционировании защищаемого приложения выделяются различные состояния, с каждым из которых ассоциируется свой набор разрешенных операций доступа.Тем самым повышается точность описания нормального поведения.

Пусть $S^r = \{S_i^r \mid i = \overline{1..k} : \exists t_r \in Bh(r), S_i^r \in t_r\}$ - множество всех допустимых состояний объекта r, которые входят хотя бы в одну из наблюдавшихся трасс поведения объекта. При этом $S^r \subseteq N_r$, т.е. построенное таким образом (в результате анализа трасс нормального поведения) множество нормальных состояний вложено во все множество допустимых состояний для приложения, но не обязательно равно ему. Пусть так же $A_i^r = \{a_{ij}^r\}$ - множество нормальных переходов (т.е. входивших хотя бы в одну из трасс нормального поведения) объекта r из состояния S_i^r в состояние S_j^r. Каждое такое множество переходов из состояния в состояние образованно множеством операций доступа, переводивших объект из одного состояния в другое. Множество всех таких операций доступа, "наблюдавшихся" во вможестве нормального поведения A^r (именно это множество строится во время профилирования приложения в AppArmor) есть объеденение всех множеств таких операций доступа по всем состояниям: $A^r = \bigcup_i A_i^r$. Очевидно, что если существуют два состояния i и j такие, что $A_i^r \neq A_j^r$, то либо $A_i^r \subset A^r$, либо $A_j^r \subset A^r$, либо верны оба строгих вложения. Содержательно это значит, что множество всех наблюдавшихся операций доступа A^r шире, чем по крайней мере одно из множест допустимых операций доступа для одного из состояний приложения. Иными словами, существующий в AppArmor подход предоставляет приложению (по крайней мере в некоторых его состояниях) больше привилегий, чем реально необходимо для корректной работы. Это происходит из-за того, что AppArmor не различает состояний приложения.

Модификация подхода AppArmor позволяет частично устранить эту проблему, позволяя различать состояния приложения и ограничивать приложение минимальными привилегиями, необходимыми для корректной работы в текущем состоянии. При этом поведение защищаемого приложения описывается в терминах операций доступа, наблюдение за которыми позволяет так же установить факт смены приложением его состояния без потребности в модификации исходного кода приложения.

Предложенная модификация возможна в тех случаях, когда можно представить поведение защищаемой программы в виде конечного автомата с достаточно небольшим пространством состояний (десятки). Напомним, что нам не доступны исходные коды программ и мы не имеем возможности модифицировать программы, а исходная информация о нормальном поведении представлена в виде множества трасс выполнения в терминах операций доступа. В этой ситуации, тем не менее, мы можем описать функционирование защищаемой программы в виде конечного автомата для класса сетевых сервисов, реализующих известные сетевые протоколы. Это обусловлено двумя факторами:

- имеется экспертное знание о логике работы таких программ, в том числе осуществляемых ими операциях доступа во время штатного функционирования,
- количество внутренних состояний таких программ не велико (ограниченно десятками состояний), что делает описание функционирования в виде конечного автомата целесообразным.

Модифицированная процедура генерации профиля будет выглядеть при этом следующим образом:

1. Выбранное для профилировки приложение запускается достаточное количество раз, аналогично стандартной для AppArmor процедуре. Критерий достаточности является вопросом методологии тестирования (рассматривается в том числе самими авторами ApArmor) и его детальное обсуждение не входит в данную работу. Сохраняются трассы выполнения приложения,
2. Посредством автоматизированной процедуры строится конечный автомат, принимающий все цепочки, входящие в наблюдавшиеся трассы выполнения программы. При этом часть наблюдавшихся действий приложения ассоциируется с состояниями конечного автомата (образуя переходы-петли), а часть – с переходами между состояниями. Построенный автомат минимизируется,
3. Администратор в ходе итеративной процедуры модифицирует полученный автомат, добавляя и удаляя новые состояния и переходы, разбивая состояния на несколько раздельных состояний с переходами между ними и модифицируя вид правил

(операций доступа), входящих в состав состояний и переходов (например обобщая объекты файловой системы с помощью регулярных выражений),

4. Полученный конечный автомат сохраняется в виде профиля. В данном профиле образуются подпрофили двух видов: именованные состояния и переходы между состояниями, каждый из которых включает в себя набор соответствующих правил. Одно из состояний помечается как начальное.

Полученный профиль может быть в дальнейшем отредактирован стандартным для AppArmor способом (т.е. ручное редактирование администратором). Модифицированная процедура обнаружения отклонений от нормального поведения выглядит следующим образом:

1. При запуске приложения с данной версией приложения ассоциируется переменная, хранящая текущее состояние приложения. Текущим состоянием приложения полагается начальное,
2. При попытке защищаемого приложения осуществить некоторую операцию доступа, проверяется, входит ли соответствующее правило в набор правил из текущего состояния. Если входит – то действие разрешается,
3. Если действие не разрешается набором правил текущего состояния, то последовательно проверяются переходы из текущего состояния. Если действие разрешается набором правил, входящим в переход из текущего состояния, то оно разрешается, а текущее состояние приложения меняется результирующее состояние перехода,
4. Если действие не разрешается ни одним из правил, входящим в наборы правил для переходов из текущего состояния, то приложение получает отказ в доступе и соответствующая запись заносится в системный журнал.

Заключение

В рамках данной работы была предложена модификация подхода к описанию нормального поведения и обнаружению отклонений от него, существующего в AppArmor. Предложенная в модификации процедура профилировки приложения (включающая в себя сбор трасс нормального поведения в терминах операций доступа и описание нормального поведения в виде конечного автомата), а так же процедура обнаружения отклонений от нормального поведения, позволяют повысить точность описаний нормального поведения приложений. Данная модификация применима для класса приложений, реализующих известные сетевые протоколы (т.е. сервисов, логика работы которых

известна). Данный подход может быть адаптирован так же для системы SELinux. Для практической реализации планируется именно адаптация предложенного подхода к системе SELinux. Результатом такой адаптации будут:

- Возможность создания более точных (в смысле точности описания разрешенных для приложения операций доступа) политик SELinux,
- Автоматизированное средство для создания правил SELinux, защищающих отдельные приложения.

Литература

1. M. Roesch. Snort: Lightweight intrusion detection for networks // In Proceedings of the USENIX LISA conference, November 1999.

2. H. Debar, M. Dacier, A. Wespi. Towards a taxonomy of intrusion detection systems // Computer Networks 1999, pp. 805-822, 1999.

3. Stefan Axelsson. Intrusion Detection Systems: A Survey and Taxonomy // Dept. of Computer Engineering, Chalmers University of Technology, Gotenborg, Sweden, 2000.

4. K. Wang and S. J. Stolfo, "Anomalous Payload-based Network Intrusion Detection", Proceedings of Recent Advance in Intrusion Detection, France, September 2004.

5. Bolzoni, D., Zambon, E., Etalle, S., Hartel, P.: POSEIDON: a 2-tier Anomaly-based Network Intrusion Detection System. In: IWIA '06: Proc. 4th IEEE International Workshop on Information Assurance, IEEE Computer Society Press (2006) pp. 144–156

6. Гамаюнов Д.Ю. Обнаружение компьютерных атак на основе анализа поведения сетевых объектов // ВМиК МГУ, Москва, 2007.

7. POSIX capabilities // [www] http://www.gentoo.org/proj/en/hardened/capabilities.xml

8. Michael Wikberg. Secure computing: SELinux // [www] http://www.tml.tkk.fi/Publications/C/25/papers/Wikberg_final.pdf, 2007.

9. Runge, C. (2004), SELinux: A new approach to secure systems, Technical report, Red Hat, Inc.

10. Redhat SELinux Guide // [www] http://www.redhat.com/docs/manuals/ enterprise/ RHEL-4- Manual/selinux-guide

11. AppArmor Detail // [www] http://en.opensuse.org/AppArmor_Detail, 2006.

12. R. Spenneberg. Shutting out Intruders with AppArmor // [www] http://www.linux-magazine.com/w3/issue/69/Shutting_out_Intruders_with_AppArmor.pdf, 2007.

Кирюшин М.В., Бахмуров А.Г.

Исследование методов и разработка средств сжатия и распаковки трасс

Введение

При проектировании и исследовании различных технических систем активно применяется *моделирование*, то есть создание и анализ *модели* исследуемой системы Моделью называют некоторую упрощенную систему, адекватно представляющую интересующие исследователей свойства исходной системы.

Важным на практике методом моделирования является имитационное моделирование, при котором моделирующая программа воспроизводит структуру исследуемой системы и функционирование её компонентов.

В ходе имитационного эксперимента поведение исследуемой системы либо непосредственно отображается системой моделирования (например, в системе AnyLogic [1]), либо сохраняется в виде трассы имитационной модели для последующего анализа различными инструментами (как, например, в стенде моделирования бортового вычислительного комплекса [2]). Такие трассы могут занимать очень большой объем дискового пространства (единицы и десятки гигабайт). Отсюда возникает задача сжатия трасс имитационных моделей. Под сжатием будем понимать некоторое преобразование входного потока, уменьшающее его размер, для которого существует обратное преобразование, позволяющее получить в точности исходные данные. Будем рассматривать два варианта задачи сжатия трасс:

- Сжатие, ориентированное на хранение трасс. В этом случае при работе с трассой будет необходимо распаковывать ее полностью
- Сжатие, позволяющее работать с сжатой трассой без распаковки или распаковывая только ее часть.

В рамках данной работы приведено решение первого варианта задачи, а также предлагается идея решения второго. Возможности сжатия трасс имитационных моделей исследуются применительно к стенду математического моделирования комплекса бортового оборудования (СММ КБО) [2].

1. Формат трассы стенда СММ КБО

Трасса событий представляет собой совокупность нескольких файлов, в каждом из которых содержится последовательность записей, описывающая выполнение имитационного эксперимента. Имеются следующие файлы:

- Основная трасса событий;
- Расширенная трасса событий;
- Трасса групп событий, записи которой описывают логические связи между событиями;
- Трасса состояний, в которой записаны состояний компонентов модели в ходе эксперимента.

В данной работе мы рассмотрим сжатие основной трассы.
В основной трассе в хронологическом порядке записаны записи о событиях, произошедших в имитационной модели в ходе эксперимента. Событие описывается следующей структурой языка Си:

```
struct event_data {
      long long EvTime; // время возникновения
      события (в микросекундах)
      short Proc;          // номер компонента,
      в котором произошло событие
      short  Oper;         // номер оператора в
      тексте модели, соответствующего событию
      short  SubType;      // подтип события
      char   Type;         // тип события
      unsigned long long extlink; // смещение
      в файле
      внешней трассы (отсутствует у некоторых
      типов событий)
       ...
}
```

Основные атрибуты события – время возникновения (EvTime), компонент, в котором произошло событие (Proc) и его тип (Type). Также у событий некоторых типов могут быть дополнительные атрибуты, которые также записаны в структуре event_data. Например, extlink – смещение в файле внешней трассы, по которому находятся дополнительные данные, связанные с событием.

2. Метод сжатия

Исследовав классические алгоритмы сжатия [5] (RLE, LZ, LZW, Huffman, арифметическое сжатие и др.), и проведя серию экспериментов с архиваторами, использующими эти алгоритмы, мы поняли, что на них нельзя получить хорошую степень сжатия, так как данные алгоритмы довольно универсальны, а файлы трасс содержат данные, записанные в заранее известном формате. Поэтому на таких специфических данных, как трассы имитационных моделей, можно добиться лучшего качества сжатия либо путем изменения существующих алгоритмов сжатия или создания новых, либо путем *предобработки* трассы – изменения формы представления данных, записанных в трассе так, чтобы эффективность сжатия традиционными универсальными архиваторами улучшилась. В ходе данной работы было принято решение выбрать второй вариант.

Очевидно, что чем "однороднее" данные в файле, тем лучше он будет сжиматься, так как для словарных методов вхождения подстрок в файл чаще будет заменяться на индекс в словаре, для статистических методов сжатия получим небольшое количество часто встречающихся символов, а для алгоритмов, использующих предсказание [3], оно будет сбываться чаще, чем на неоднородных данных. Исходя из сказанного, была сделана попытка преобразовать трассу так, чтобы данные, записанные в ней, были представлены в как можно более однородном виде. Рассмотрим один из способов, как можно это сделать.

Трасса представляет собой последовательность записей, а каждая запись – набор атрибутов. Разобьем трассу на несколько потоков, в каждом из которых записана последовательность какого-то одного атрибута события трассы. Например, для основной трассы получим последовательности временных меток (Time), номеров компонентов, в которых произошло событие (Proc), типов и подтипов событий (Type, SubType). Затем к некоторым из последовательностей применим дополнительные преобразования, которые устранят избыточность и позволят улучшить последующее сжатие.

Для каждого типа события создадим отдельный поток, в котором будет храниться последовательность значений дополнительных атрибутов этого типа события. Также в отдельный поток вынесем последовательность значений атрибута Extlink, упорядоченную в порядке появления событий в трассе. Если в каком-то типе события присутствует несколько дополнительных параметров, то для каждого из них создадим отдельный поток. Таким образом получаем, что в каждом потоке записана последовательность значений только одного какого-то атрибута события. Этот процесс показан схематически на Рис. 1.

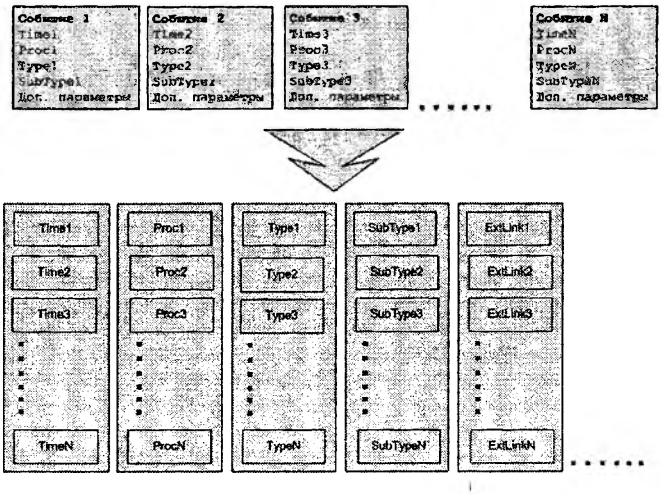


Рисунок 1. Общая схема преобразования трассы

К некоторым из таких последовательностей были применены дополнительные преобразования с целью уменьшения избыточности данных и улучшения качества сжатия. Одно из них описано в [4]. Рассмотрим кратко эти преобразования. Чтобы уменьшить объем, занимаемый последовательностью значений атрибута, была предпринята попытка сократить количество байт, отданное под хранение одного элемента последовательности. Например, в событии на метку времени отводится 8 байт, но зачастую можно обойтись меньшим объёмом. Для сжатия меток времени был использован один из вариантов RLE кодирования.

В последовательности меток времени RLE кодирование было применено после того, как из каждой метки времени в последовательности вычли предыдущую. Благодаря этому в последовательности стало много подряд идущих нулей (т.к. в один момент времени часто происходит много событий), что способствует хорошей степени сжатия алгоритмом RLE.

Для последовательностей остальных атрибутов также были предприняты попытки сделать их более однородными – хранить разность между соседними значениями вместо самих значений, убрать из последовательности неопределенные значения (равные -1), применить RLE .

3. Результаты экспериментов и выводы

С целью оценки эффективности работы созданного программного средства была проведена серия экспериментов с трассами из различных проектов.

На тех же трассах с целью исследования возможности сжатия стандартными архиваторами и оценки качества сжатия, предоставляемого ими, была проведена серия экспериментов на архиваторах gzip 1.3.5, bzip2 1.0.3 и rar 3.70 с ключами командной строки -9, -9, m5 соответственно (максимальная степень сжатия)

Проверочные трассы перечислены в таблице 1. С результатами экспериментов можно ознакомиться в таблицах 2 - 5

Таблица 1 – Проверочные трассы[1]

№	Проект	Эксперимент	Размер
1	Struna	Pohod88	15.3Mb
2	Struna	Very Big Trace	1.2Gb
3	DemoLang	Test600	184Mb
4	Struna	Kingstown	152Mb

Проект Struna – модель корабельного навигационного комплекса, содержит 97 компонентов, в экспериментах №№ 1,4 часть компонентов была отключена. Проект DemoLang – демонстрационная модель, 20 компонентов. Время работы моделируемого комплекса во всех экспериментах было равно 10 мин.

Таблица 2 – Результаты экспериментов (коэффициент сжатия)

№	RAR	PP[2]+RAR	Bzip2	PP+bzip2	Gzip	PP+gzip
1	28	218	6.12	318.75	7.2	193.1
2	10.5	215	5.3	324.3	3.4	135.1
3	15	454	5.5	420	5.3	221.4
4	9.8	100	5.8	138.1	5.5	69.7

Таблица 3 - Результаты экспериментов(объем сжатой трассы)

№	RAR	PP+RAR	Bzip2	PP+bzip2	Gzip	PP+gzip
1	545.8Kb	66.9Kb	2.5Mb	47.9Kb	2.1Mb	79.2Kb
2	114Mb	5.58Mb	225Mb	3.7Mb	349.1Mb	8.88Mb
3	12.2Mb	404.7Kb	33.1Mb	438.0Kb	34.7Mb	831.1Kb
4	15.5Mb	1.5Mb	26.0Mb	1.1Mb	27.2Mb	2.18Mb

[1] В столбце "Размер" указан суммарный размер основной трассы, трасс состояний и групп событий.

[2] PP – предобработка перед применением архиватора.

Таблица 4 – Результаты экспериментов (время сжатия)

№	RAR	PP+RAR	bzip2	PP+bzip2	Gzip	PP+gzip
1	2.6sec	2.0sec	3.0sec	6.5sec	0.9sec	2.2sec
2	4m 37sec	4m 38sec	5m 41sec	8m 53sec	2m 22sec	2m 44sec
3	39.0sec	31.6sec	38.8sec	1m 38sec	17.7sec	35.6sec
4	29.4sec	18.6sec	45.7sec	36.4sec	12.7sec	18.3sec

Таблица 5 – Результаты экспериментов (время распаковки)

№	RAR	PP+RAR	bzip2	PP+bzip2	Gzip	PP+gzip
1	2.2sec	0.9sec	1.7sec	2.1sec	0.7sec	2.3sec
2	1m 27sec	2m 29sec	2m 25sec	3m 9sec	1m 3sec	2m 31sec
3	10.6sec	21.0sec	26.8sec	23.2sec	12.3sec	20.7sec
4	10.5sec	16.2sec	20.5sec	22.8sec	9.1sec	18.4sec

Таким образом, применение предобработки трассы позволяет достичь степени сжатия в среднем в 200 раз и увеличить ее относительно универсальных архиваторов в 10 – 20 раз. Архиватор bzip2 сжимает обработанную трассу в среднем в 300 раз, rar - в 245, gzip – в 154 раза. Но, к сожалению, время сжатия и распаковки при этом увеличивается на некоторых архиваторах в среднем на 50%.

Перспективы развития предложенного метода следующие:

- Модификация существующих методов преобразования трассы. Один их вариантов такой модификации – применение предикторов, описанных в [3]. Этот метод основан на том, что при проходе трассы вычисляются несколько функций (предикторов) от уже обработанных событий. И если следующее событие или атрибут события равен значению одного из предикторов, то в упакованную трассу записывается 1, иначе – 0 и само событие или атрибут. Если хорошо подобрать функции-предсказатели, то этим методом можно добиться хороших результатов.
- Реализация возможности поиска (выборки) событий в заданном промежутке времени по сжатой трассе, либо посредством распаковки только какой-то ее части. Возможный вариант такой реализации – разбить трассу на блоки одинакового размера, сжать их по отдельности и проиндексировать по времени события. А по запросу распаковывать только те блок(и), в котором содержатся искомые события

Литература

[1] Система моделирования AnyLogic [HTML] http://www.xjtek.ru/anylogic/

[2] Грибов Д.И., Смелянский Р.Л., Комплексное моделирование бортового оборудования летательного аппарата // Методы и средства обработки информации. Труды второй всероссийской научной конференции. – М.: Издательский отдел факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова, 2005. – С.59-74

[3] Martin Burtscher, Metha Jeeradit., Compressing Extended Program Traces Using Value Predictors // Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques. 2003. P. 167-176. [PDF] (http://ieeexplore.ieee.org/iel5/8771/27774/01238012.pdf)

[4] Eric E. Johnson, Jiheng Ha PDATS - Lossless Address Trace Compression for Reducing File Size and Access Time//IEEE International Phoenix Conference on Computers and Communications. 1994. P. 213-219 [PDF] (ftp://tracebase.nmsu.edu/pub/pubs/pdats.pdf)

[5] Ватолин Д, А. Ратушняк, М. Смирнов, В. Юкин., Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. М.: Диалог-МИФИ, 2003. 384 с.

Стеван Шчепанович, Естественно-математический факультет Университета Черногории

АНАЛИЗ КОМПЬЮТЕРНОЙ СИСТЕМЫ КАК УЗЛА КОММУТАЦИИ В СЕТЯХ ДЛЯ ПЕРЕДАЧИ ДАННЫХ

Резюме

В данной работе использован метод имитационного моделирования для описания функционирования компьютерной системы как узла коммутации. В рамках модели изучаются взаимоотношения аппаратной и системно-программной структуры узла, с одной стороны, и коммуникационных программ обслуживания пакетов, с другой стороны. Сделана попытка предоставить принципиально новое описание функционирования коммуникационной программы для обслуживания пакета в узле коммутации. На основании анализа поведения данных программ оцениваются характеристики работы узла коммутации и компьютерной сети в целом.

В целях разработки рабочей нагрузки модели, дополнительно разработан метод, который основывается на поведении коммуникационных программ и независимости этого поведения от аппаратной и системно-программной структуры. Такой подход позволяет в рамках единой системы испытать как алгорифмовые, так и временные характеристики моделирования.

1. Введение

В современных сетях по переносу данных, узлы коммутации реализовываются как компьютерные системы специального назначения, и обозначены как маршрутизаторы (*routers*). Исследования показывают, что ограничивающим фактором повышения эффективности узла коммутации (маршрутизатора) является именно оптимальный источник их архитектуры, т.е. их элементарные структуры и составляющие компоненты. Это актуализирует проблемы разработки новых методов и средств анализа узла коммутации на этапе их проектирования, что бы позволило оценить их эффективность в кратчайшие сроки, с меньшими затратами и более высокой точностью, чем сейчас. В данной работе предложен метод, который преодолевает недостатки существующих

методов анализа и оценки работы узла и на них базирующихся компьютерных сетей.

Компьютерная система, которая создает узел коммутации имеет сложную структуру. Ее динамичное поведение характеризуют различные нелинейные феномены. Причины нелинейности находятся в алгоритмах, используемых для управления памятью, распределения процессов, нахождения и исправления ошибок, а так же и в факте, что проток движения через узел варьирует разными и часто непредсказуемыми способами. Прямое экспериментирование над реальным узлом требует дополнительных финансовых затрат, а может проводиться только на завершающем этапе проектирования и установки данного коммуникационного устройства. Поэтому, в этой работе, мы сделали цифровой анализ влияния внутренней структуры и рабочей нагрузки на работу узла и испытали изменения в движении, которые из-за этого происходят. Целью работы было предложит дешевый, достаточно точный, эффективный и оперативный цифровой анализ и проектирование основного элемента пакетных сетей, т.е. узла коммутации или маршрутизатора.

2. Формулировка задания анализа узла коммутации и предпосылки по применению метода

Задание оценки работы узла коммутации может быть задано путем моделирования процессов, которые происходят в этой компьютерной системе, и может быть сформулирован следующим образом: Задание для узла коммутации (специализированная компьютерная система) R как комплекс аппаратных и системно-программных компонентов, и задана коммуникационная программа K_p, которая обслуживает пакеты, поступившие в узел. Нужно найти способ оценки работы узла коммутации R на заданной программе K_p без использования самого узла R или его прототипа.

Узел коммутации, который является предметом анализа, это компьютерная система специального назначения. Она имеет: процессор (один или более), оперативную память и память для буфера пакетов, флешь память для хранения оперативной системы, коммуникационных программ и конфигурационных папок и порты коммуникационных каналов входа/выхода. Узел коммутации работает под управлением своей оперативной системы.

В рамках той же сети маршрутизаторы могут междоусобно сотрудничать по обмену релевантными информациями о состоянии и топологии сети. Эти информации нужны коммуникационным программам в ходе выбора наилучшего пути для направления пакетов. Сотрудничество между процессами, выполняющимися на разных

маршрутизаторов, чаще всего осуществляется путем обмена сообщениями (служебных пакетов).

Рабочую нагрузку (*payload*) системы составляют направления пакетов, поступающих в узел, и обслуживаются деятельностью коммуникационных программ. Рабочую нагрузку характеризует случайность появления и асинхронное влияние.

3. Декомпозиция компьютерной системы как узла коммутации

В целях анализа и исследования имеет смысл сделать декомпозицию компьютерной системы, который работает как узел коммутации. В соответствии с этим, внутри самой компьютерной системы идентифицированы три основные целостности:

- ***Физическая или аппаратная структура***, которая состоит из аппаратных элементов системы (процессоры, оперативная или буферная память, интерфейсы входных/выходных коммуникационных каналов). Это ресурсы, которыми пользуются программные процессы при выполнения своих действий по перенаправлению пакетов.
- ***Рабочая среда***, т.е. активное окружение, которое создают процессы оперативной системы узла. Рабочая среда строит логический интерфейс между коммуникационными процессами и физической структурой и обеспечивает выполнение сервисных функций маршрутизатора. Это среда в которой выполняются коммуникационные процессы.
- ***Рабочая нагрузка***, которая состоит из направления пакетов, содержит характеристики этих пакетов (длину, тип, приоритет) и описание очередности, в которой они входят в узел.

4. Неформальный анализ заданий и определение математической модели

При определении заданий для оценки рабочих характеристик, предполагается, что отсутствует коммуникационный узел R или его прототип. Тогда для оценки его рабочих характеристик Y нужна модель F системы R, которая установит отношение $Y = F(X)$, т.е. **математическую модель наблюдаемой системы.** В общем случае, совокупность X является параметрами и входными переменными узла коммутации, т.е. его аппаратно - программную структуру и его рабочую нагрузку.

Пока физическая структура и поведение оперативной системы с небольшими отступлениями постоянны в течение одной серии испытаний, рабочая нагрузка, т.е. направления пакетов меняются, и соответственно, меняется и поведение аппликационных программ. Поэтому, при определении модели F с достаточной точностью можно взять, что X, математическими средствами, описывает поведение коммуникационных процессов из совокупности K_p.

Система уравнения $Y = F(X)$, это математическая модель, которая описывает функционирование узла коммутации посредством математических операторов и объектов. Таким образом, анализ реального узла меняется на исследование его математической или цифровой модели. Если эти операции, при помощи соответствующих алгоритмов, реализуются на компьютере, тогда речь идет о имитационном моделировании.

5. Системно независимая мера компьютерной работы коммуникационной программы

Ключевой вопрос, задаваемый при моделировании компьютерных систем, это: можно ли на основании результатов наблюдения за поведением коммуникационной программы в некой экспериментальной компьютерной среде, прогнозировать ее поведение в другой, аналитической компьютерной среде, т.е в реальном узле коммутации?

Для ответа на этот вопрос нужно решить следующие проблемы:

- Доказать ***наличие таких действий***, которые могут быть свойственными всем исполнительным средам в которых может функционировать наблюдаемая коммуникационная программа. К таким действиям относится и обслуживание пакетов в узле коммутации.
- Уметь ***предвидеть последовательность действий*** коммуникационного процесса в конкретной (анализируемой) структуре, отличая ее архитектуру от архитектуры экспериментальной компьютерной среды и последовательности деятельности коммуникационного процесса в экстремальной среде.
- Определить ***объем компьютерной работы***, не зависящий от физических характеристик конкретной компьютерной системы, а, одновременно позволяющий эффективную интерпретацию на конкретную компьютерную систему. Такой объем компьютерной работы обозначен, как ***системно независимый объем программы*** или ***инварианта поведения программ*** [2] (*program behavior Invariant*).

6. Логические ресурсы узла коммутации

Для определения системно независимой меры компьютерной работы введено понятие логический ресурс. Это – средство управления обработкой данных, конкретно – обслуживанием пакетов, поступающих в узел коммутации. Логические ресурсы в виде услуг коммуникационным процессам предлагает исполнительная, т.е. системно-программная среда, которая, опять же использует физические или аппаратные ресурсы.

Взаимосвязь логического ресурса с аппаратной структурой компьютерной системы выражается при помощи таких терминов, как: связь (*linking*) и время связи. Под связью логического ресурса подразумевается сравнение обращения к логическому ресурсу с реализацией этого ресурса в терминах физической среды. Другими словами, связь это процесс превращения программы в форму, удобную для ее исполнения в выбранной физической среде.

Если поведение коммуникационной программы опишется при помощи ряда событий, характеризирующих использование логических ресурсов, тогда ряд таких событий дает системно независимое описание поведения программы, которая хранит все информационные и управленческие взаимосвязи [3]. Это описание системно независимо в том смысле, что в любой среде с наличием в ней соответствующего набора логических ресурсов, коммуникационный процесс выдает единый ряд обращений к нему.

Важно напомнить о том, что уровень логического ресурса не зафиксирован. Его выбор определяют цели анализа, и, конечно, уровень детализации до которого рассматривается поведение программы.

7. Создание модели развития узла коммутации

Модель развития узла коммутации играет фундаментальную роль в оценке его рабочих возможностей. Модель содержит все доступные информации о структуре узла и его поведении, которые, как таковые, необходимы для создания имитационной модели. В процессе создания модели развития формируется замысел модели. Основное содержание данного этапа это переход от общего описания узла к его математической модели, т.е. процесс формализации. Этот этап, тем не менее "геуристический", т.е. исследователь в основном руководствуется только совей интуицией, опираясь на имеющиеся знания и понимание процесса работы узла. Чем глубже познание конкретного узла, тем лучше будет модель развития, и легче и успешнее будет его перенос в имитационную модель.

Имитационная модель узла коммутации должно описывать не только его статическую структуру, но и его динамическое поведению Понятия динамической структуры связаны с изменением состояния коммуникационного процесса во времени. Время в модели, или моделирующее время реального узла означается как системное время. Это время не совпадает с техническим временем, в котором в компьютере происходит процесс моделирования. Есть два основных метода задачи времени в модели, т.н.з. системное время.

В первой модели задается постоянный временный шаг Δt. Показатель Δt имеет большое влияние на процесс моделирования таким образом, что любое положительное наращивание времени относительно реального события останавливает процесс моделирования в его естественном движении вперед. Данным методом экономится техническое время в процессе моделирования, но при неправильном выборе временного шага можно получить ошибочные результаты. В другом методе время меняется дискретно, при этом задается шаг до следующего события. Событие это момент обращения коммуникационного процесса логическому ресурсу.

Если поведение модели во времен в сути соответствует поседению узла коммуникации согласно некоторым условиям отношений между разными аспектами модели и узла, у нас есть имитационная модель.

8. Создание имитационной модели узла коммутации

Имитационная модель узла коммутации описывает работу узла коммутации в виде ряда операций, выполняемых на каком-либо компьютере. Неотъемлемой частью имитационной модели является описание объектов, которые формируют данную компьютерную систему (узел) и описание структуры системы, т.е. совокупностью связей между объектами. Модель реализуется как программа на основании отражения соответствующей функциональной структуры или модели развития в программную структуру, приспособленную под исполнение на компьютере. Имитационное моделирование (*simulation*) это выполнение цифрового эксперимента на компьютере путем выполнения этой программы на какой-то группе входных данных.

Имитационная модель, разработанная в данной работе, это программная система на языке C^{++}, который и по структуре и по функции похож на наблюдаемую компьютерную программу. Она (модель) состоит группы независимо развитых программных модулей. Коме этого, использован и библиотечный подход при создании имитационной модели. Она заключается в разделении технологического

цикла создания и пользования моделью на несколько этапов. Первый этап моделирования заключается в создании библиотеки моделей существующих аппаратных и системно-программных компонентов наблюдаемого узла коммутации. С этой целью созданы модели: модель протокола физического стандарта *IEEE 802.3*; модель протокола транспортного уровня *TCP* и т.д.

В рамках предложенной модели создано три типа объектов:

- ***Динамичные объекты*** системы, представляющие коммуникационные процессы, которые обслуживают поступающие пакеты. Они поддерживают динамическое поведение узла.
- ***Пассивные объекты*** системы, представляющие аппаратные и системно-программные ресурсы системы, т.е. исполнителей, на которых выполняется данная программа для коммутации пакета в узле.
- ***Наблюдательный*** (мониторный, инструментальный) объекты.

9. Проверка соответствия и калибровки модели

Основной вопрос, возникающий при использовании имитационной модели: насколько хорошо модель представляет соответствующую реальную компьютерную систему? Это вопрос адекватности модели. Проблема доказывания адекватности модели в основном сводится на демонстрацию корректности создания модели и правильность структуры и функциональности модели, т.е. достаточно ли хорошо поддерживают структуру и функциональность модели. В данной работе использован ***реалистичный подход*** к созданию адекватного имитационного модуля.

Результатам, полученным при моделировании, нельзя верить, если модель недостаточно точная. Проблема точности модели видна в том факте, что результат моделирования точны только в том случае, когда идентичные процессы происходят в модели и в реальной системе. Точность модели определяется при помощи рабочих показателей, выбранных для оценки системы. Модель достаточно точная, если разница между данными рабочих показателей, полученных при моделировании узла коммутации и данные, измеренные в реальной системе, находятся в пределах разрешенной ошибки.

В течение процесса оценки точности имитационной модели, рассматривается: внутреннее состояние модели, сочетание с реальной системой и правильность чтения и толкования результатов. Если точность модели неудовлетворительная, тогда модель должна быть изменена, а процесс проверки повторен. Такая операция называется калибровкой модели. Цель калибровки заключается в снижении или

устранении неточностей, которые могут возникнуть при формировании модели. Между тем, и точная модель может давать неправильные результаты даже в процессе калибровки, если метод решения, или показатели входных параметров, или и одно и другое неисправные.

10. Применение предложенного метода

Метод, разработанный в данной работе, использован для анализа экспериментального узла коммутации на Естественно-математическом факультете в Подгорице. Целью анализа этого узла коммутации была разработка и исследование разных конфигураций его аппаратной и системно-программной структуры, а также общая оценка качества использованных проектных решений коммутационных программ и использование компьютерной сети. Для достижения данной цели потребовалось:

- Разработать соответствующую имитационную модель, которая адекватно представит компьютерную систему, которая работает как узел коммутации;
- Обеспечить модульный принцип создания имитационной модели и структурный способ оценки полученных результатов, для облегчения самого процесса имитационного моделирования и обеспечения условий для его развития;
- Разработать экспериментальный узел коммутации, который представительно представит данную систему, и который послужит как контрольный узел для проверки адекватности и правильности модели;
- Разработать и проанализировать средства оценки эффективности функционирования данного контрольного узла и использовать их как входные параметры имитационной модели.

Архитектура контрольного узла максимально открыта для расширения и изменения конфигурации системы. Это означает, что аппаратно-программные компоненты этой системы и способ их объединения, выбраны так, что было возможно:

- Простое изменение структуры и конфигурации аппаратуры и системного софтвера,
- Изменение стратегии управления способом коммутации полностью и способом распределения пакетов, использованных в программной поддержке их подсистем,

- Максимальная возможность выбора оптимального уровня детальности модели с точки зрения анализа системы, характеристик программы и точности замеров.

11. Модель рабочей нагрузки

Для симуляции рабочей нагрузки узла коммутации в ходе процесса анализа, использовалась разработанная для этого программа. Эта же программа использовалась и в реальном экспериментировании на узле коммутации, а также для симуляции, т.е. имитационном эксперименте с моделью. Причиной этому была проверка исправности модели с целью вынесения правильных выводов о работе наблюдаемой системы. При создании выше указанной специальной программы учитывалась его много системная независимость, т.к. только в таком случае при помощи этой программы можно получить хорошую оценку рабочих показателей наблюдаемого узла коммутации.

Параметры и переменные рабочей нагрузки, представленные в имитационной модели, могут быть определены как ***стохастические*** при помощи соответствующих распределений случайных чисел или как ***детерминистические*** на основании искусственных информаций и данных, собранных в ходе исполнения помянутой специальной программы на контрольном узле. В течение процесса анализа наблюдаемого узла коммутации, а для получения разных информаций о его работе, использовались оба способа формирования рабочей нагрузки.

12. Результаты моделирования

В ходе реального, а также и имитационного эксперимента, реализовано исследование рабочих показателей контрольного узла коммутации, сделанного на основании персонального компьютера с микропроцессором *Pentium* 4. Большая часть этого анализа посвящена анализу времени задержки пакета в узле и анализу скорости передачи пакета через узел.

Наблюдалось и поведение узла в условиях разной нагрузки. Перенос информаций между абонентами сети, которые обслуживает данный узел, осуществляется в пакетах. Длина пакета переменная и ограничена на 257 байтов. Каждый пакет, кроме полезных данных, несет и адрес направления. Пакет принимает узел коммуникации и размещает его в свою память. Данные переносятся в режиме дуплекс. В этой связи, соединения между узлами содержат два коммуникационных канала, по одному для каждого направления передачи.

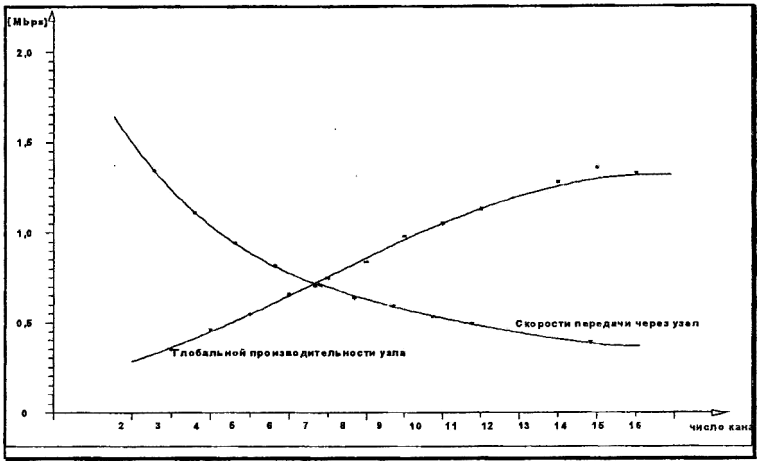
Результаты моделирования контрольного узла получаются как среднее случайных показателей, таких как: среднее количество пакетов в системе, средняя задержка, производительность узла и т.д. Результаты, полученные при моделировании, показывают, что рабочие показатели узла в значительной мере зависят от трех переменных: числа активных каналов, интенсивности входного течения и длины пакета.

Замер рабочих показателей контрольного узла выполнено при помощи так называемого программного монитора, установкой дополнительных инструкций и счетчика внутри сетевой оперативной системой узла. Как системный таймер использован таймер компьютерной системы узла. Снимали только движение между двумя определенными абонентами, при чем оба абонента и слали и получали данные. Остальные абоненты работали в системе в интерактивном режиме. Сделано семь экспериментов с разным числом дополнительных активных абонентов. Число активных абонентов не менялось в ходе выполнения полной серии эксперимента. Полученные результаты показали, что этим методом можно оценить рабочие показатели узла коммутации с точностью между 45 и 60 процентами.

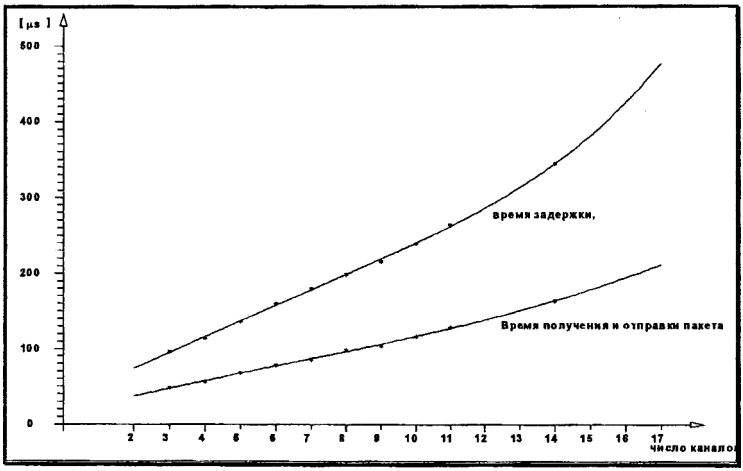
Диаграммы зависимости внешних параметров узла от числа активно подключенных каналов получены таким способом, что по каждому каналу передавалось Пуассоново течение (*Poisson*) одной и той же интенсивности. Эти диаграммы представлены на следующих изображениях. Интенсивность прохода по каналу составляла 90 пакета в секунду, что означает, что средний интервал времени между проходом двух пакетов составлял в среднем 0,011 секунды.

Предпосылка, что длина приходящих пакетов случайных размеров, которые подчиняются законам равномерного распределения и движутся в границах между 3 и 257 байтами. Каждый отдельный имитационный эксперимент для какого-то определенного числа каналов, проводился до тех пор, пока не заканчивалось обслуживание 2500 пакетов. Число не принятых пакетов не превышало 1,3 процента от общего числа наблюдаемых пакетов, а в большинстве экспериментов этот процент не превышал 0,5.

На изображении показаны зависимости скорости передачи через узел и производительности узла от количества активно подключенных каналов. Кривая глобальной производительности узла показывает, что с ростом количества активно подключенных каналов свыше тринадцати, глобальная производительность узла начинает падать. Это объясняется появлением насыщения в узле.



Нижнее изображение показывает, что время задержки, как и время получения и отправки пакета, растет при появлении насыщения в узле коммутации. Такие результаты, конечно, были ожидаемыми, с той разницей, что в практике появление насыщения замечается четь раньше, уже при подключении десяти каналов.



В том примере предполагалось, что по всем каналам проходит течение одинаковой интенсивности, а места получений пакетов определялись по случайному выбору. Такие предпосылки – достаточно грубая аппроксимация, но мы показали, что этим методом, быстрым и

простым способом можно получить полезные информации о функционировании узла коммутации.

Реально, что подтверждают и имитационные эксперименты, внешние параметры узла это функция не одной, а двух, трех и более переменных, таких как: число активных каналов, длина пакета, интенсивность входа и т.д. Также, показывает существование и весьма выраженной взаимозависимости внутренних параметров узла, что значительно усложняет анализ. В дальнейших исследованиях один из двух ключевых внутренних параметров держался постоянным, а остальные два варьировали.

В следующих примерах использовались два метода полного факторного планирования для статистической обработки полученных результатов. Таким образом, получаем следующие функциональные зависимости релевантных рабочих показателей узла от числа подключенных каналов k и длины пакетов d.

• время приема пакета в узел:

$$y_1 = d(1{,}4 \cdot 10^{-1} - 2{,}1 \cdot 10^{-4} \cdot d + 5{,}3 \cdot 10^{-2} k + 3 \cdot 10^{-5} d \cdot k)\,[\mu s]$$

• время отправки пакета из узла:

$$y_2 = d(1{,}6 \cdot 10^{-1} - 1{,}6 \cdot 10^{-4} \cdot d + 4 \cdot 10^{-2} \cdot k + 10^{-4} d \cdot k)\,[\mu s]$$

• время ожидания освобождения канала выдачи:

$$y_3 = 1{,}6 \cdot 10^{-1} - 8{,}3 \cdot 10^{-5} \cdot d + 4{,}2 \cdot 10^{-2} \cdot k + 5{,}8 \cdot 10^{-5} \cdot d \cdot k\,[\mu s]$$

• время задержки пакета в узле коммутации :

$$y_5 = 2{,}7 \cdot 10^{-1} + y_1 + y_2 + y_3\,[\mu s]$$

• скорость передачи определена с помощью оптимального метода, известного как ортогональное планирование второго ряда:

$$y_6 = 2{,}1 + 7{,}6 \cdot 10^{-3} d - 1{,}1 \cdot 10^{-5} d^2 - 2{,}4 \cdot 10^{-1} k + 1{,}2 \cdot 10^{-2} k^2 - 1{,}3 \cdot 10^{-4} d \cdot k\,[Mbps]$$

13. Заключение

Этим анализом узла коммутации имитационной системы моделирования получены ответы на следующие вопросы:

- Оценена производительность узла проектируемого узла коммутации,
- Оценена пропускной способность коммуникационных каналов, проходящих через узел,
- Оценена нагрузка отдельных компонентов, входящих в состав узла (процессора, модулей памяти, сетевых интерфейсов и т.п.), как и эффективность, их использования,

- Выполнена верификация коммуникационных протоколов,
- Установлены возможные потенциальные застои и конфликты алгоритмов и системы в целом и т.д.

В данной работе предложен дешевый, достаточно точный, эффективный и оперативный цифровой метод для анализа и проектирования компьютерной системы как узла коммутации. Данная методология может использоваться и в разработке крупных проектов, когда невозможно собрать на одном месте все компоненты проектируемой компьютерной сети и, когда она еще не установлена.

Литература

[1] Смелянский Р.Л.: *"Методы анализа и оценки производительности последовательных вычислительных систем"*. // Методическая разработка., Москва: Ротапринт НИВЦ МГУ, 1990.

[2] Смелянский Р.Л.: *"Об инварианте поведения программ"*. В журнале: Вестник Московского университета Серия 15. Вычислительная математика и кибернетика, No 4., 1990., с 54-58.

[3] Смелянский Р.Л.: *"Анализ производительности распределенных микропроцессорных систем на основе инварианта поведения программ"*. Диссертация на соискание учёной степени доктора физико-математических наук. Москва: МГУ, 1991.

[4] Шчепанович С., Леонтьев Д., Мратов А.: *"Вопросы определения калибровочных параметров имитационных моделей серверных приложений"*. Программные системы и инструменты., Тематический сборник № 1. Москва: Изд-во факультета ВМиК МГУ, 2000., с. 48-55.

Аннотации

Балаханов В.А., Кокарев В.А. Муравьиный алгоритм построения статико-динамических расписаний и исследование его эффективности // Программные системы и инструменты. Тематический сборник, М.: Изд-во факультета ВМиК МГУ, 2008. стр. 118 –126//

В работе приводится описание муравьиного алгоритма построения статико-динамического расписания. В качестве примера такой задачи рассматривается задача построения расписания для однопроцессорных систем реального времени, работающих в соответствии со стандартом ARINC-653. В работе приведены результаты исследования эффективности разработанного алгоритма и ее зависимости от некоторых характеристик исходных данных и параметров алгоритма. Также приведены результаты исследования адаптационных механизмов муравьиных алгоритмов на примере рассматриваемой задачи.

Библ.: 7 назв.

Балашов В.В., Бахмуров А.Г., Волканов Д.Ю., Смелянский Р.Л., Чистолинов М.В., Ющенко Н.В. Применение стенда полунатурного моделирования для разработки вычислительных систем морского навигационного комплекса. // Программные системы и инструменты. Тематический сборник № 9, М.: Изд-во факультета ВМиК МГУ, 2008. стр. 153–165//

В статье описывается структура стенда полунатурного моделирования, разработанного в лаборатории вычислительных комплексов факультета ВМК МГУ, предлагается технология комплексирования ВС РВ с применением стенда и описываются изменения в стенде и приведён опыт применения стенда для оценки архитектуры ВС РВ морских навигационных комплексов.

Ил.: 1 рис., Библ.: 7 назв.

Балашов В.В., Шестов П.Е.

// Программные системы и инструменты. Тематический сборник № 9, М.: Изд-во факультета ВМиК МГУ, 2008. стр. 166–178//

В составе встроенных систем реального времени широко используются каналы с централизованным управлением и статической схемой построения расписания информационного обмена. При построении расписания может быть выявлена несовместимость требований к информационному обмену. В этом случае невозможно построить расписание, удовлетворяющее всем требованиям. В работе предложен алгоритм автоматического формирования рекомендаций по обеспечению совместимости требований к информационному обмену

по каналу с централизованным управлением. Приведены результаты экспериментального исследования алгоритма.

Библ.: 8 назв.

Брусенцов Н.П. О содержательном истолковании логико-алгебраических выражений // Программные системы и инструменты. Тематический сборник № 9. М.: Изд-во факультета ВМиК МГУ, 2008. стр. 127–128//

На примере непарадоксального необходимого следования представлено содержательное истолкование логико-алгебраических выражений.

Доложено на Ломоносовских чтениях 2008 г. на факультете ВМиК МГУ.

Библ.: 2 назв.

Владимирова Ю.С. Силлогистика Аристотеля и гераклитов принцип сосуществования противоположностей // Программные системы и инструменты. Тематический сборник № 9. М.: Изд-во факультета ВМиК МГУ, 2008. стр. 129–132//

Рассматриваются условия непарадоксальности рассуждений на примере аристотелевой силлогистики, представленной в алгебре совокупностей.

Доложено на Ломоносовских чтениях 2008 г. на факультете ВМиК МГУ.

Библ.: 7 назв.

Гамаюнов Д.Ю., Сапожников А.В. Обнаружение аномального поведения приложений на уровне ядра операционной системы // Программные системы и инструменты. Тематический сборник № 9, М.: Изд-во факультета ВМиК МГУ, 2008 стр. 179–192//

Данная работа посвящена задаче описания нормального поведения приложений на узлах защищаемой системы и обнаружения отклонений от нормального поведения. В работе рассматриваются подходы, применяемые в системах SELinux и AppArmor. Для подхода, применяемого в AppArmor, предлагается модификация, основанная на формальной модели описания поведения сетевых объектов и позволяющая повысить точность описаний нормального поведения приложений за счет учета внутренних состояний профилируемого приложения.

Библ.: 12 назв.

Гришин С.В., Ватолин Д.С., Лукин А.С., Путилин С.Ю., Стрельников К.Н., Обзор блочных методов оценки движения в

цифровых видео сигналах. // Программные системы и инструменты. Тематический сборник № 9, М.: Изд-во факультета ВМиК МГУ, 2008 стр. 50–62//

В статье представлен обзор современных алгоритмов блочной оценки движения в цифровых видео сигналах, а также приведено описание общего понятия информации о движении в цифровом видео. В рамках обзора предложена классификация блочных алгоритмов оценки движения на несколько групп: шаблонные методы, методы иерархического поиска, методы, использующие наборы кандидатов, и комбинированные подходы. Для каждой группы алгоритмов описаны наиболее интересные ее представители с указанием их достоинств и недостатков.

Ил.:4. Библ.: 12 назв.

Гудков А.В., Разработка интерпретатора Лисп-подобного языка для исследования алгоритмов распределенного поиска. // Программные системы и инструменты. Тематический сборник № 9, М.: Изд-во факультета ВМиК МГУ, 2008. стр. 89–94//

В работе делается краткий обзор поисковой модели и основных алгоритмов распределенного поиска. Далее, вводится расширенный набор операторов языка Лисп в терминах рассматриваемой модели. Показывается, как с помощью представленного языка выразить запрос для любого из представленных алгоритмов. Показывается, что с его помощью возможно объединение поисковых структур в иерархию, используя комбинацию базовых алгоритмов; приведен пример. Указываются свойства и особенности языка.

Библ.: 3 назв.

Ельцин А.В., Учебные материалы в интеллектуальной обучающей системе. // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2008. стр. 133–142//

В настоящей статье рассматривается реализация авторской поддержки подготовки учебного материала для интеллектуальной обучающей системы. Интеллектуальность системы состоит в применении в обучении концептуальной способности учащегося – понимания. Отсюда следуют решения по выделяемой в авторском курсе сложной структуре, необходимой для анализа учебного материала в соответствии с применением генетического метода обучения ГРОМ (Герменевтики и Развивающего обучения Мастер).

Ил.: 1

Библ.: 6 назв.

Срджан Кадич, Предраг Станишич, естественно-математический факультет Университета Черногории. Алгоритм проверки сериализуемости выполнения множества транзакций// Программные системы и инструменты. Тематический сборник № 9, М.: Изд-во факультета ВМиК МГУ, 2008. стр. 143–152//

В работе предложен новый алгоритм проверки сериализуемосты выполнения множества транзакций. Предложенный алгоритм позволяет увеличить возможности механизма для управления транзакциями.

Библ.: 7 назв.

Илл. 4

Кирюшин М.В., Бахмуров А.Г. Исследование методов и разработка средств сжатия и распаковки трасс. // Программные системы и инструменты. Тематический сборник № 9, М.: Изд-во факультета ВМиК МГУ, 2008. стр. 193–199//

Данная работа посвящена исследованию методов и разработке средства сжатия/распаковки трасс имитационных моделей (на примере стенда полунатурного моделирования бортовых вычислительных комплексов). Созданы метод и программное средство сжатия трасс, основанное на предварительной обработке трассы перед применением распространённых архиваторов общего назначения. Проведены эксперименты с использованием реализованного средства, степень сжатия увеличилась в 10-30 раз по сравнению с «обычными» архиваторами. Предложены пути дальнейшего развития: улучшение существующего метода, реализация произвольного доступа к сжатой трассе.

Библ.: 5 назв.

Куликов Д.Л., Временной метод маскирования искажений в видео на основе обработки оптического потока. // Программные системы и инструменты. Тематический сборник № 9, М.: Изд-во факультета ВМиК МГУ, 2008 стр. 63–73//

В статье предлагается новый временной метод маскирования искажений в видео. Он основывается на подходе с использованием оптического потока и поиска неоднородностей, что позволяет более точно проводить реконструкцию векторов движения в неизвестной области, что в результате приводит к более высокому визуальному качеству маскирования. Приводятся результаты объективного и субъективного сравнения предложенного метода и аналогов. Данный метод предназначен в основном для задач постобработки видео для удаления артефактов декодирования, но может также применяться и для задач улучшения визуального качества путем удаления царапин, пятен и других дефектов из видеопотока.

Ил.:8. Библ.: 10 назв.

Мальковский М.Г., Старостин А.С. Синтаксический анализатор Treevial. Постановка задачи синтаксического анализа (текств на естественном языке) // Программные системы и инструменты. Тематический сборник № 9, М.: Изд-во факультета ВМиК МГУ, 2008. стр. 95–108//

Статья посвящена описанию синтаксического анализатора Treevial, осуществляющего анализ текстов на естественном языке. Этот анализатор является основным компонентом системы поддержки проектирования лингвистических процессоров Treeton, которая разрабатывается на кафедре алгоритмических языков факультета ВМиК МГУ, начиная с 2005 года. При создании анализатора был использован обобщенный подход к представлению синтаксических структур и правил. Он позволяет использовать анализатор как для работы с деревьями зависимостей, так и для работы с системами непосредственных составляющих, а также для работы со смешанными моделями. Авторы фокусируются главным образом на математической модели анализатора: вводятся понятия синтаксической структуры, мультиструктуры и синтаксического правила. Описывается процедура применения правила к мультиструктуре. С использованием введенных терминов формулируется задача синтаксического анализа.

Библ.: 6 назв.

Попова Е.А. Алгоритм построения карт электрической активности мозга на основе обработке сигнала ЭЭГ и его применение для исследования нейрофизиологических проблем восприятия. Тематический сборник № 9, М.: Изд-во факультета ВМиК МГУ, 2008. стр. 74–88//

Статья посвящена описанию алгоритмов построения пространственно-усредненных карт активности нейронных дипольных источников, и анализу результатов применения данных алгоритмов для обработки экспериментальных данных при исследовании нейрофизиологических проблем визуального восприятия. В работе предлагается новый подход для оценки функционального состояния мозговой деятельности, основанный на кластерном анализе временных последовательностей параметров активных зон для группы сигналов ЭЭГ.

Ил: 0

Библ.: 5 назв.

Рябов Г.Г., Кодирование состояний марковских цепей в динамике примитивных триангуляций R^3 и R^4. // Программные системы и инструменты. Тематический сборник № 9, М.: Изд-во факультета ВМиК МГУ, 2008. стр. 7–19//

В предлагаемой статье рассматриваются случайные «перестройки» (flips) примитивной триангуляции в $\mathbf{R}^3$ (с вершинами $\mathbf{Z}^3$) как марковские цепи и исследуются их свойства периодичности, разложимости и эргодичности, тем самым устанавливается асимптотическое поведение триангулированного пространства в целом. Предложены близкие методы для примитивных триангуляций в $\mathbf{R}^4$.

Илл. – 10,

Библ.: 11 назв.

Смелянский Р.Л., Шалимов А.В., Разработка системы компактного представления программ. // Программные системы и инструменты. Тематический сборник № 9, М.: Изд-во факультета ВМиК МГУ, 2008. стр. 20–31//

В статье описана система компактного представления последовательных программ, реализующая метод компактного представления программ на основе частотных характеристик их поведения. Приведены результаты теоретических и экспериментальных исследований.

Ил.: рис.: 3, табл., Библиогр.: 12 назв.

Старостин А.С., Синтаксический анализатор Treevial. Алгоритм восходящего синтаксического анализа с памятью, работающий под управлением эвристической функции. // Программные системы и инструменты. Тематический сборник № 9, М.: Изд-во факультета ВМиК МГУ, 2008. стр. 109–117//

Статья посвящена описанию алгоритма работы синтаксического анализатора Treevial, осуществляющего анализ текстов на естественном языке. Этот анализатор разрабатывается на кафедре алгоритмических языков факультета ВМиК МГУ, начиная с 2005 года. При создании анализатора был использован обобщенный подход к представлению синтаксических структур и правил. Он позволяет использовать анализатор как для работы с деревьями зависимостей, так и для работы с системами непосредственных составляющих, а также для работы со смешанными моделями. Обсуждаемый алгоритм обладает двумя важными свойствами. Во-первых, все строящиеся гипотезы о синтаксическом строении предложения динамически оцениваются с помощью эвристической штрафной функции. Во-вторых, алгоритм построен таким образом, что результаты анализа частей входного предложения могут использоваться повторно в ходе перебора (у алгоритма есть «память»).

Библ.: 7 назв.

Стеван Шчепанович, Анализ компьютерной системы как узла коммутации в сетях для передачи данных. // Программные системы и инструменты. Тематический сборник № 9, М.: Изд-во факультета ВМиК МГУ, 2008. стр. 200–212//

В данной работе использован метод имитационного моделирования для описания функционирования компьютерной системы как узла коммутации. В рамках модели изучаются взаимоотношения аппаратной и системно-программной структуры узла, с одной стороны, и коммуникационных программ обслуживания пакетов, с другой стороны. Сделана попытка предоставить принципиально новое описание функционирования коммуникационной программы для обслуживания пакета в узле коммутации. На основании анализа поведения данных программ оцениваются характеристики работы узла коммутации и компьютерной сети в целом.

Библ.: 4 назв.

Сутырин П.Г., Мальковский М.Г. Иерархические конечные автоматы и звездная высота регулярных выражений // Программные системы и инструменты. Тематический сборник № 9, М.: Изд-во факультета ВМиК МГУ, 2008. стр. 32–49//

Предлагается алгоритм преобразования конечного автомата в регулярное выражение. При анализе автомата выделяется иерархия компонент, отвечающих за звездную высоту получаемого выражения. В некоторых случаях алгоритм строит выражение наименьшей высоты для данного языка. Предложено направление для нового, конструктивного решения задачи о звездной высоте.

Библ.: 9 назв.

Научное издание

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 9

Под общей редакцией
чл.-корр. РАН Л.Н. Королева

Подготовка оригинал- макета:
Трусова Н.

Напечатано с готового оригинал-макета

Издательство ООО "МАКС Пресс"
Лицензия ИД N 00510 от 01.12.99 г.
Подписано к печати 17.12.2008 г.
Формат 60x90 1/16. Усл.печ.л. 13,75. Тираж 100 экз. Заказ 778.

119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова,
2-й учебный корпус, 627 к.
Тел. 939-3890, 939-3891. Тел./Факс 939-3891.