МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
им. М.В. ЛОМОНОСОВА

ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

ТРУДЫ
ФАКУЛЬТЕТА ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ
И КИБЕРНЕТИКИ

№ 3

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

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

Москва
2002

УДК 519.6+517.958
ББК 22.19
П75

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

Программные системы и инструменты: Тематический сборник
П75 факультета ВМиК МГУ им. Ломоносова: № 3.
/ Под ред. Л.Н. Королева. – М: Издательский отдел факультета ВМиК МГУ (лицензия ИД №05899 от 24.09.2001г.), 2002. - 224с.

В данный сборник включены научные работы и сообщения по различным темам, связанным с имитационным моделированием и синтезом систем передачи данных, машинной графикой, распознаванием образов, с проблемами распараллеливания вычислений. В его состав включены также работы алгоритмизации вычислений булевых функций и другие сообщения.

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 2002 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 5-89407-149-6

СОДЕРЖАНИЕ

СБОРНИК

" Программные системы и инструменты"

От редактора

В предлагаемом читателю тематическом сборнике публикуются статьи, посвященные в основном описаниям инструментальных программных систем, разработанных авторами публикаций.

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

Большинство статей представляют собой тексты докладов, прочитанных на Ломоносовских Чтениях 2002 на факультете ВМиК.

Перечень разделов сборника №3 таков:

- общие вопросы программирования и информатики;
- имитационное моделирование и синтез систем передачи информации;
- машинная графика и обработка образов;
- распараллеливание вычислений;
- сообщения;

Статьи сборника будут интересны специалистам, разрабатывающим прикладные программные системы и использующим новые информационные технологии.

Л.Н. Королев

Раздел I
Общие вопросы программирования и информатики

Брусенцов Н.П., Владимирова Ю.С.

Троичное конструктное кодирование булевых выражений

Булева алгебра – первооснова, или начало, всякой науки, исследующей взаимосвязи, будь то логика, математика, информатика или, скажем, структурная лингвистика. Ввиду же фундаментальности наук этого рода не будет преувеличением усмотреть в ней и начало науки вообще – способности рассуждать, умозаключать, доказывать.

Компьютерная информатика также «произрастает» из булевой алгебры, что наиболее очевидно. Однако дальнейшее развитие способности компьютера, становление «искусственного интеллекта», оставляет желать лучшего. Впрочем, при нынешнем засилии формализма и естественный человеческий интеллект оказывается под угрозой – поневоле превращаемся в роботов.

Пора бы придать компьютерной информатике здравый диалектический характер.

В этой статье речь пойдет об усовершенствованной конструктной реализации булевой алгебры в диалоговой системе структурированного программирования ДССП [1-3].

Конструктами в ДССП называются нестандартные, определяемые (конструируемые) пользователем типы данных, введением которых достигается высокоуровневая специализация этой системы в заданном классе приложений. Комплекс программ, обеспечивающий возможность декларирования конструктных переменных и реализующий базисный набор операций конструкта, называется, поддерживающим этот конструкт *конструктивом* [4]. Специализация и развитие ДССП в нужном направлении осуществляется дозагрузкой соответствующих конструктивов.

Конструкты типа «булево выражение» предоставляют возможность оперирования переменными, принимающими в качестве значений *n*-арные выражения булевой алгебры, и функционально полный набор базисных операций, позволяющих в сочетании с штатными средствами конструирования программ в ДССП эффективно реализовать логико-алгебраические процедуры [5].

Функционально конструкт определяется форматом принимаемых переменными значений и набором интерпретирующих

эти значения базисных операций. Формат характеризуется структурой, информационной емкостью и способами доступа. Например, формат «вектор битов» с параметром n – длина вектора предоставляет доступ к отдельным битам по их номерам, а также к вектору в целом. Вектор битов надлежащим выбором базисных операций можно интерпретировать как двоичное число без знака, либо со знаком, как целое, либо дробное и т. п. Но тот же вектор битов можно интерпретировать как n-арную элементарную конъюнкцию (либо дизъюнкцию), приняв в качестве базисных операций побитные инверсию, конъюнкцию и дизъюнкцию.

Принцип отображения булевых выражений конструктами в простейших случаях заключается во взаимно однозначном сопоставлении входящих в выражение букв-переменных (следуя Аристотелю, будем называть их *терминами*) последовательно пронумерованным компонентам-битам в формате конструкта. Другими словами, каждая переменная представлена в формате конструкта собственным битом. Значение же, принимаемое битом, указывает статус его переменной в отображаемом выражении. Например, элементарная конъюнкция $xyz'u$ отображается вектором 1101 при условии, что неинвертированному термину сопоставлена цифра 1, а инвертированному – цифра 0.

Заметим, что всевозможные 2^n n-арные элементарные конъюнкции пронумерованы значениями отображающего n-битного вектора, интерпретируемыми как двоичные натуральные числа. В приведенном примере номера последовательно убывают от 1111 для $xyzu$ до 0000 для $x'y'z'u'$.

Эта нумерация использована в конструкте, отображающем булевы выражения в совершенной дизъюнктивной нормальной форме (СДНФ). Его формат – 2^n-компонентный вектор битов, пронумерованных n-битными числами от 11...1 до 00...0, и таким образом однозначно сопоставленных n-арным элементарным конъюнкциям. Биты, соответствующие входящим в отображаемое СДНФ-выражение конъюнкциям, принимают значение 1, а все прочие – значение 0. Например, при $n = 2$ отображающий вектор состоит из 4-х битов, пронумерованных числами 11, 10, 01, 00, которым соответствуют элементарные конъюнкции xy, xy', $x'y$, $x'y'$, так что выражение $xy \vee x'y$ отобразится в 1010, а выражение $xy' \vee x'y \vee x'y'$ отобразится в 0111.

Конструкт описанного типа, названный *двоичной ДК-шкалой*, позволяет эффективно компьютеризовать алгебру n-арных СДНФ-выражений. Очевидно, что операции конъюнкции и дизъюнкции над такими выражениями сводятся к побитным конъюнкциям и дизъюнкциям ДК-шкал, а операция отрицания-дополнения СДНФ-

выражения – к побитной инверсии его ДК-шкалы. Существенное достоинство ДК-шкалы – ее экономность: произвольная n-арная булева функция кодируется 2^n битами.

Выражения в совершенной конъюнктивной нормальной форме (СКНФ-выражения) аналогично отображаются n-арной двоичной КД-шкалой. Вернее, 2^n-битный вектор допускает ДК-интерпретацию и КД-интерпретацию: ДК – это дизъюнкция элементарных конъюнкций (СДНФ), а КД – конъюнкция элементарных дизъюнкций (СКНФ).

Наряду с ДК- и КД-шкалами не лишены смысла и менее экономные конструкты на основе формата *цепь* элементарных конъюнкций либо дизъюнкций. Цепь – это совокупность n-арных векторов, допускающая добавление, удаление, а также перестановку отдельных ее членов. В отличие от шкалы, где членам СНФ-выражения сопоставлены позиции битов-компонент отображающего вектора, так что номера позиций кодируют соответствующие члены, в цепи коды членов СНФ-выражения содержатся непосредственно и могут располагаться в любой последовательности, в частности, упорядочиваться по тому или иному критерию, что способствует дальнейшему развитию конструктной алгебры.

Шкалы и цепи, основанные на отображении элементарных конъюнкций и дизъюнкций векторами битов (двоичные конструкты), позволяют компьютеризовать алгебру совершенных нормальных форм, СДНФ- и СКНФ-выражений. В системе с двоичными конструктами эффективно реализуются процедуры синтеза и преобразования СНФ-выражений, такие как тождественное преобразование выражения в двойственную форму, инвертирование, получение дополнения, получение дуала (выражения, двойственного данному), получение единого выражения из нескольких заданных по предписанным взаимосвязям, выявление и доказательство отношений, в которых состоят сопоставляемые выражения, решение булевых уравнений [5, 6].

Компьютеризация булевой алгебры в полном объеме достигнута применением конструктов, в основу которых положен вектор трехзначных элементов (тритов). Конструкты этого рода естественно называть троичными. О том, что в троичном коде успешно преодолевается несовершенство двоичного кодирования убедительно свидетельствует троичный симметричный код чисел, в котором три значения трита интерпретируются как 1, 0, -1 т. е. к двоичным 1 и 0 добавлена отрицательная единица. Не имеющих удовлетворительного решения в двоичном коде проблем представления чисел со знаком и округления чисел в симметричном троичном коде просто нет.

Точно так же компенсируется неполноценность отображения вектором битов элементарных конъюнкции и дизъюнкции, оказывающаяся причиной того, что двоичные конструкты отображают

булевы выражения только в совершенных нормальных формах. Используемые в них для представления элементарных конъюнкций и дизъюнкций n-битные векторы способны кодировать только индивидные конъюнкции и предполные дизъюнкции, но не могут отобразить элементарную конъюнкцию или дизъюнкцию, в которой некоторые термины умалчиваются («элиминированы» по Булю-Порецкому), т. е. в которой статус терминов трехзначен: неинвертированное вхождение, вхождение под знаком инверсии, невхождение.

Сопоставив этим состояниям значения трита 1, -1, 0, которые далее ради удобства будем обозначать +, -, 0, получаем отображение n-тритным вектором не только индивидных конъюнкций и предполных дизъюнкций, но и элементарных n-арных конъюнкций и дизъюнкций произвольного вида. Так, конъюнкциям $xyz'u$, $xz'u$, yz', z' будут соответствовать значения 4-тритного конструкта-вектора К-типа: ++-+, +0-+, 0+-0, 00-0, а дизъюнкциям $x' \vee y' \vee z \vee u'$, $x' \vee z \vee u'$, $y' \vee z$, z – значения 4-тритного конструкта-вектора Д-типа: --+-, -0+-, 0-+0, 00+0.

Построенные на n-тритных векторах ДК- и КД-цепи при должным образом пересмотренных наборах интерпретирующих базисных процедур способны отображать теперь не только СНФ-выражения, но и произвольное булево выражение в нормальной форме с фиксированным порядком размещения терминов в элементарных конъюнкциях и дизъюнкциях. Например, выражение $xy \vee x'y$, отобразимое 2-арной двоичной ДК-цепью 11 01, троичной 2-арной цепью отобразимо как в СДНФ ++ -+, так и в минимальной ДНФ 0+, а выражение
$xy' \vee x'y \vee x'y'$ (двоичная ДК-цепь 10 01 00) троичной ДК-цепью отображается в четырех вариантах: +- -+ --, +- -0, 0- -+, -0 0-, -0 0-, т. е. в СДНФ, в двух тупиковых и в минимальной ДНФ.

Угадывается «алгебраическая полнота» троичного отображения, и в булевой алгебре она действительно имеет место: посредством троичных констуктов удается перепоручить компьютеру практически все, что может делать в булевой алгебре человек и даже то, чего еще не может (например, минимизации произвольного булева выражения [7]).

Описанное усовершенствование компьютерной реализации булевой алгебры представляет собой один из результатов конструктного подхода к информатике. Результат фундаментальный, поскольку касается основы и вместе с тем открывает пути совершенствования последующих ступеней – алгебраизации силлогистики, модальной и диалектической логики [8], упорядочения теории вероятностей и нечетких множеств [9, 10].

Литература

1. Брусенцов Н.П., Златкус Т.В., Руднев И.А. ДССП-диалоговая система структурированного программирования // Программное оснащение микрокомпьютеров. – М.: Изд-во Моск. ун-та, 1982. С. 11-40.

2. Брусенцов Н.П. Микрокомпьютеры. – М.: «Наука», 1985. С. 141-170.

3. Развиваемый адаптивный язык РАЯ диалоговой системы программирования ДССП / Н.П. Брусенцов, В.Б. Захаров, И.А. Руднев, С.А.Сидоров, Н.А. Чанышев. – М.: Изд-во Моск. ун-та, 1987. 80 с.

4. Концептуальная характеристика РИИИС-процессора / Н.П. Брусенцов, С.П. Маслов, Х. Рамиль Альварес, С.А. Сидоров // Интегрированная система обучения, конструирования программ и разработки дидактических материалов. - М.: Изд-во ф-та ВМиК МГУ, 1996 г. С. 16-43.

5. Владимирова Ю.С. Конструктная реализация булевой алгебры // Интегрированная система обучения, конструирования программ и разработки дидактических материалов. - М.: Изд-во ф-та ВМиК МГУ, 1996 г. С. 44-69.

6. Брусенцов Н.П., Владимирова Ю.С. Решение булевых уравнений // Методы математического моделирования. – М.: Диалог-МГУ, 1998. С. 59-68. Solution of Boolean Equations. // Computational mathematics and modeling, Vol. 9 , № 4, 1998, pp. 287-295.

7. Брусенцов Н.П., Владимирова Ю.С. Троичный минимизатор булевых выражений // Программные системы и инструменты. Тематический сборник № 2. – М.: Факультет ВМиК МГУ, 2001. С. 205-207.

8. Брусенцов Н.П. Трехзначная диалектическая логика // Программные системы и инструменты. Тематический сборник № 2. – М.: Факультет ВМиК МГУ, 2001. С. 36-44.

9. Брусенцов Н.П., Деркач А.Ю. Логическая модель теории вероятностей и нечетких множеств Заде // Цифровая обработка информации и управление в чрезвычайных ситуациях. Материалы Второй международной конференции (28-30 ноября 2000 г., Минск.) – Минск: Институт технической кибернетики НАН Беларуси, 2000. Том 1, с. 41-44.

10. Брусенцов Н.П., Деркач А.Ю. Трехзначная логика, нечеткие множества и теория вероятностей // Программные системы и инструменты. Тематический сборник № 2. – М.: Факультет ВМиК МГУ, 2001. С. 88-91.

Брусенцов Н.П.

Упорядочение булевой алгебры

Инструментальную основу современной информатики составляет булева алгебра с базисными операциями отрицания, конъюнкции и дизъюнкции, определенными таблицами, в которых набор значений как операндов, так и результатов операций исчерпывается двумя, интерпретируемыми обычно как "истина" и "ложь" и обозначаемыми соответственно цифрами "1" и "0", либо буквами: русскими "И", "Л", латинскими "T", "L". Это алгебра двухзначной "логики высказываний", отождествляющей символ операции отрицания "¬" с частицей *"не-"*, символ конъюнкции "∧" - с грамматическим союзом *"и"*, символ дизъюнкции "∨" - с *"или"*. Истолковываемые таким образом символы операций называют *логическими связками.*

Выражения булевой алгебры представляют собой составные высказывания, "истинность" которых вычислима посредством определяющих связки таблиц. Применение же связок к выражениям и выявление действующих при этом алгебраических законов приводит к *исчислению высказываний.* Впрочем, последнее предпочитают формулировать с привлечением неэлементарной связки - *материальной импликации*, сопоставляемой отношению следования в естественных языках, которому она, увы, в сущности не тождественна. Таким образом, получилось исчисление, необыкновенно благоприятное для исследователей парадоксов, но на практике более чем бесполезное, ибо не соответствует здравому смыслу.

Каноническая, обходящаяся без импликации, булева алгебра семантическими парадоксами не омрачена, но и ее постулаты не вполне адекватны реальности и здравомыслию. Вследствие принятого в этой алгебре "априорного" закона исключенного третьего вне ее пределов оказались и модальности, и аристотелева силлогистика, иными словами, и интуиционизм, и диалектика.

Что-то в булевой алгебре не так, как должно быть по логике бытия. Прежде всего подозрение падает на операцию булева отрицания, которая почему-то определена так, что порождает *дополнение* отрицаемого до рассматриваемого многообразия возможностей (до *универсума*), т.е. в смысле "все то, что не А", где А - отрицаемое. Но ведь по здравому смыслу *отрицанием* данной определенности признается любая не совместимая с ней (противоречащая ей) определенность из имеющихся в универсуме. И ясно, что отрицание составных(неэлементарных) определенностей неоднозначно. Например,

отрицаниями простейшей составной определенности xy - *конъюнкции* (совместности) элементарных определенностей x и y - будут: x', y', $x'y$, xy', $x'y'$, $x' \vee y'$.

В булевой алгебре из этих шести принято последнее, *отрицание-дополнение*. Оно "замечательно" тем, что не оставляет места промежуточному третьему: все, что не отрицается, утверждается и обратно, так что алгебра булевых выражений сохраняет свойственную элементарным определенностям двухзначность. Но оно не столь элементарно как диаметральная *инверсия* $x'y'$, сводящаяся к инвертированию в отрицаемом выражении всех вхождений каждой элементарной определенности, каждого термина.

Условимся обозначать дополнение префиксом "$\neg$", а инверсию - постфиксом "штрих". Инверсия и дополнение, например, конъюнкции xy выразятся так:

$$(xy)' \equiv x'y', \qquad \neg(xy) \equiv xy' \vee x'y \vee x'y' \equiv x' \vee y'$$

В отличие от инверсии как потерминного инвертирования выражения, отрицание-дополнение достигается исключением отрицаемого выражения xy из характеризующей универсум полной дизъюнкции $xy \vee xy' \vee x'y \vee x'y' \equiv 1$. Затем полученное СДНФ-выражение минимизируется в $x' \vee y'$, оказывающееся *двойственным* (дуалом) тому, что дала инверсия. Таким образом, булево дополнение $\neg e$ выражения e есть дуал его инверсии e':

$$\neg e \equiv \delta(e')$$

Операция получения двойственного (дуала) δ состоит во взаимозамене в выражении-операнде символов $\wedge$ на $\vee$ и $\vee$ на $\wedge$.

Данное соотношение обратимо - инверсия в свою очередь есть дуал дополнения:

$$e' \equiv \delta(\neg e).$$

Однако инверсия, как уже было сказано, элементарней дополнения. Это видно из того, что функциональную полноту булевой алгебры обеспечивает пара дополнение-конъюнкция, либо же пара дополнение-дизъюнкция, тогда как в системе с инверсией необходимы и конъюнкция, и дизъюнкция. Дело в том, что непременный в определении дополнения закон исключенного третьего $e \vee \neg e \equiv 1$ на инверсию не распространяется и поэтому тождество де Моргана

$$x \wedge y \equiv \neg(\neg x \vee \neg y),$$

определяющее посредством дополнения конъюнкцию через дизъюнкцию и обратно, в случае инверсии не имеет места. В силлогистике инверсии соответствует контрарность, а дополнению - контрадикторность [1].

Базисные операции алгебры, поименованные и обозначенные каждая собственным знаком (функтором), определены, с одной стороны, чисто формально таблицами истинности и предписаниями механических манипуляций, производимых над последовательностями символов, отображающими алгебраические выражения. С другой стороны, эти операции имеют содержательную интерпретацию (истолкование, смысл, семантику), и не одну.

Выражения булевой алгебры обычно истолковывают как *высказывания* - предложения, принимающие одно из двух значений "истинности", или как атрибуты классов, к которым относится либо, наоборот, не может относиться классифицируемая по представленным элементарными терминами *критериям* (признакам) вещь. С классами связана *объемная* (*экстенсиональная*) интерпретация выражений, согласно которой подклассы называют содержащимися во включающих их классах и подчиненными им, что составляет диаметральную противоположность принятой в естественном рассуждении *смысловой* (*интенсиональной*) интерпретации. Интенсионально атрибут класса *содержится* в атрибутах его подклассов и *подчинен* каждому из них, *сказывается* о каждом, *необходимо присущ* каждому включенному в него, *необходимо следует* из него.

Рассмотрим конкретные примеры интенсиональной интерпретации простейшего булева выражения - элементарной конъюнкции. В n-терминном универсуме, т.е. при различении по n первичным критериям, имеем n-местную (n-арную) конъюнкцию, каждой из компонент которой придается один из трех статусов: *необходимо присущее* / *антиприсущее* / *привходящее*. Булева конъюнкция содержит (необходимо) присущие ей термины непосредственно, без каких-либо функциональных знаков, каждый антприсущий термин - под знаком инверсии, привходящие термины не содержатся (умалчиваются). Например, в трехтерминном x,y,z-универсуме конъюнкции xyz' присущи x и y, антиприсуще z, конъюнкции же xz' присуще x, антиприсуще z, а умалчиваемое в ней y - привходяще, т.е. необходимо не присуще и не антиприсуще.

Еще одна практически важная интерпретация булевых выражений - *теоретико-множественная* [2]. Та же конъюнкция xyz' истолковывается как множество, которому *принадлежат* элементы x, y и *антипринадлежит* (не может принадлежать) элемент z. Конъюнкция xz' представляет собой нечеткое множество, которому необходимо принадлежит x, антипринадлежит z, а элемент y не принадлежит и не антипринадлежит с необходимостью (безусловно). Элементы обретают привходящий статус в результате дизъюнкции множеств, например: $xyz' \vee xy'z' \equiv xz'$ [3].

Конъюнкции без умолчания терминов являются четкими, категорическими множествами в традиционном понимании. Дизъюнкции конъюнкций (ДНФ-выражения) представляют собой нечеткие множества. Множества вообще, четкие и нечеткие, условимся называть *совокупностями*. Произвольное булево выражение в теоретико-множественной (совокупностной) интерпретации истолковывается как совокупность первичных терминов. Операция инверсии выражения инвертирует представленную этим выражением совокупность терминов, превращая принадлежащие ей в антипринадлежащие, антипринадлежащие в принадлежащие, а привходящие оставляя без изменения.

К булевым выражениям как к совокупностям терминов применимы, помимо инверсии, операции пересечения, объединения и теоретико-множественной разности. Примеры:

$$(xyz')' \equiv x'y'z$$

$$xyz' \cap xz \equiv xyz' \cap (xyz \vee xy'z) \equiv xyz' \vee xy'z' \equiv xz'$$

$$xyz' \cup xz \equiv xyz' \cup (xyz \vee xy'z) \equiv xyz$$

$$xyz' \setminus xz \equiv xyz' \cap (xz)' \equiv xyz' \cap x'z' \equiv x'z'$$

Вместе с тем, базисные булевы операции (связки) - конъюнкция и дизъюнкция - обретают новое истолкование как конъюнкция и дизъюнкция совокупностей. Четкая совокупность (множество) формируется путем конъюнкции нечетких, например: $xz' \wedge xy \equiv xyz'$. Нечеткие совокупности суть дизъюнкции четких, например: $xyz' \vee xy'z' \equiv xz'$.

Совокупностная интерпретация булевых выражений естественно изоморфна их интенсиональной и экстенсиональной интерпретациям. Так, элементарная конъюнкция, не содержащая привходящих (умалчиваемых) терминов, т.е. представляющая четкую совокупность терминов, интенсионально понимается как *индивидное* в принятом универсуме понятие, а экстенсионально - как атрибут индивидного класса. Нечетким совокупностям соответствуют размытые, содержащие несущественные термины, понятия и неиндивидные классы. Нельзя, однако, отождествлять (и даже смешивать) категории класса и множества, как это принято в логике, и в традиционной, и в математической.

Посредством базисных связок - инверсии, конъюнкции и дизъюнкции - выразима произвольная n-местная (n-терминная, n-арная) *булева функция*. Более того, в нормальных формах (ДНФ и КНФ) инвертируются только первичные термины, а в совершенных нормальных формах (СДНФ и СКНФ) нет умалчивания терминов. Выражение в совершенной дизъюнктивной форме представляет собой дизъюнкцию индивидных (n-терминных) конъюнкций, определяющую

класс соответствующих этим конъюнкциям четких совокупностей (множеств) терминов. Этому классу соответствует в алгебре 2-й ступени (в булевой алгебре дизъюнктов) четкая совокупность *n*-терминных индивидов, которые в логике предикатов называют "предметами". Нечеткой совокупности таких индивидов, допускающей привходящую принадлежность ей некоторых из них, в алгебре 1-й ступени соответствует класс с привходящей включенностью в него отдельных подклассов - *нечеткий класс.*

Например, характеристическая функция отношения следования, представленного в аристотелевой силлогистике общеутвердительным суждением "Всякое x есть y" ("Всякому x присуще y", "y содержится в x", "y необходимо следует из x"), выражается конъюнкцией дизъюнктов вида:

$$\mathrm{V}xy\mathrm{V}'xy'\,\mathrm{V}x'y'$$

где знак интегральной дизъюнкции, аналога интегральной суммы, - дизъюнкт V - символизирует дизъюнкцию значений, принимаемых поддизъюнктным выражением на элементах характеризуемой совокупности, распространенную на всю эту совокупность.

Рассматриваемая нечеткая совокупность 2-й ступени характеризуется необходимой принадлежностью ей индивидов xy и $x'y'$, необходимой непринадлежностью (антипринадлежностью) xy' и привходящей принадлежностью индивида $x'y$, который в выражении умалчивается. Атрибут соответствующего этой совокупности индивидов класса в алгебре 1-й ступени определяется как общий всем ее членам, т.е. дизъюнкцией атрибутов всех необходимо принадлежащих совокупности, а также привходящих индивидов. В рассматриваемом примере такой нечеткой дизъюнкцией будет

$$xy \vee \sigma x'y \vee x'y'$$

где σ - символ третьего "значения истинности" - *привходящего:* не-0 и не-1, а нечто небезусловное, некатегоричное, оцениваемое как вероятность, либо как доля (часть, процент) дискретного значения 1: $0 < \sigma < 1$.

Эта небулева дизъюнкция выражает характеристическую функцию *импликации*, которая в двухзначной булевой алгебре огрублена в "материальную импликацию" $(x \to y) \equiv xy \vee x'y \vee x'y'$ и, вместе с тем, в "эквивалентность" $(x \leftrightarrow y) \equiv xy \vee x'y'$.

Возвращаясь к выражению нечеткой совокупности 2-й ступени - характеристической функции отношения *необходимого следования*, приведем другие выражения этой совокупности, полученные путем тождественного преобразования представленного выше ее выражения:

$$\mathrm{V}xy\mathrm{V}'xy'\,\mathrm{V}x'y' \equiv \mathrm{V}x\mathrm{V}'xy'\,\mathrm{V}y' \equiv \mathrm{V}x\Lambda(x' \vee y)\mathrm{V}y'$$

Каждое из этих выражений выявляет в отношении необходимого следования y из x ту или иную его характерную черту, позволяет, так сказать, посмотреть на него с разных сторон. Согласно первому выражению, y следует из x, если в универсуме имеются (существуют) вещи класса xy и класса $x'y'$, но не может быть вещей класса xy'. Второе выражение требует существования классов x и y', исключая существование класса xy'. В третьем выражении требование несуществования класса xy' преобразовано в удовлетворенность каждой из вещей условию материальной импликации $x \to y \equiv x' \vee y$.

То, что всеобщая удовлетворенность импликации равносильна несуществованию (т.е. исключенности) xy', еще раз указывает на несущественность в ее СДНФ-выражении члена $x'y$. А то, что удовлетворенность импликации не означает необходимого следования (кроме нее требуется существование x и существование y'), снимает проблему "строгих" и "сильных" импликаций, опровергая вместе с тем общепринятое "положение", будто бы из противоречия следует все что угодно. Импликация действительно удовлетворяется в случае противоречивости антецедента, но ведь противоречивое не существует, а из несуществующего ничто не может следовать.

Однако возвратимся к 1-й ступени, с тем чтобы уточнить и четко сформулировать принципы, положенные в ее основание. Составляя первооснову булевой алгебры, 1-я ступень является тем самым и фундаментом всех последующих, определяемых посредством этой алгебры разделов информатики - теории множеств, арифметики и теории чисел, числовых функций, а главное - искусства достоверного рассуждения (доказательства), которое по Аристотелю "будучи способом исследования, прокладывает путь к началам всех учений" [4, 101b3].

Оптимальное упорядочение информатики в целом достижимо лишь при оптимальной упорядоченности ее основания - булевой алгебры, а порядок в алгебре задается выбором ее *базиса* - совокупности первичных операций, посредством которых выражаются все прочие функции. Впрочем, базисом обычно называют минимальную функционально полную систему операций. Однако при таком понимании возникает затруднение с той же булевой алгеброй, в которой базисными операциями служат $\neg, \wedge, \vee$, а из них выделяются два минимальных функционально полных набора: $\neg, \wedge$ и $\neg, \vee$. Что же называть ее базисом?

Условимся называть базисом алгебры функционально полную, не обязательно минимальную, совокупность *элементарных* операций, применяемых в "правильных" выражениях этой алгебры. Понимаемый таким образом базис может быть минимальным либо избыточным, как

это имеет место в булевой алгебре с операциями дополнения, конъюнкции и дизъюнкции. Для упорядочения же существенно, чтобы базисные операции были элементарными, несоставными.

Элементарный неизбыточный базис булевой алгебры составляют операции *инверсии, конъюнкции и дизъюнкции.*

Операция инверсии булева выражения определена как инвертирование каждого вхождения каждого из имеющихся в этом выражении терминов. Например:

$$(x' \vee y)' \equiv x \vee y', \ (xy')' \equiv x'y, \ (xy' \vee x'y)' \equiv x'y \vee xy',$$
$$0' \equiv (xx')' \equiv x'x \equiv 0.$$

Операции конъюнкции и дизъюнкции выражений однозначно определены присущими им законами дистрибутивности и поглощения, однако в оптимально упорядоченной системе они реализуемы более просто и эффективно, например, как соответственно пересечение и объединение представляющих данные выражения совокупностей их СДНФ-членов. *Упорядочение* (синоним "структурирования") - это выявление в рассматриваемом естественного порядка взаимосвязей и неуклонное подчинение ему в процессе исследования и развития системы. Образцами упорядочения, к сожалению не получившими должного понимания и применения, являются "Упорядочение дел человеческих" и "Великая дидактика" Яна Амоса Коменского, а в наше время - "Структурированное программирование" Эдсгера Дейкстры. Основоположником упорядочения следует признать первооткрывателя диалектики Демокрита, который указал на существование всеобщего естественного миропорядка и назвал его *Логосом*. С учетом дальнейшей модификации смысла этого слова следует истолковывать его как адекватное отображение указанного миропорядка в сознании и в языке людей, в информатике.

По-видимому, фундаментальный принцип и метод упорядочения состоит в выявлении и последовательном использовании спиралеобразной иерархии компонент отображаемой взаимосвязи. Например, компонентами молекул полагаются атомы, в свою очередь сконструированные из компонент атомного уровня, которые затем декомпануются на более элементарные составляющие, и т.д. Хотя можно ведь "слепить" молекулу и непосредственно из элементарных частиц. В "неструктурированном программировании" именно так и делали, пока Э. Дейкстра не выступил с призывом "Разделяй и властвуй", впрочем, нажлежащего понимания не получившим.

В булевой алгебре структурирование состоит в том, что выражения, отображающие функции терминов-переменных,

конструируются не непосредственно из терминов и базисных связок, а в виде иерархии, на нижнем уровне которой порождаются стандартные подвыражения, используемые в качестве компонент, связываемых друг с другом базисными связками на следующем уровне. Наглядный пример естественного структурирования выражений - совершенные нормальные формы: дизъюнктивная и конъюнктивная.

Совершенная дизъюнктивная нормальная форма (СДНФ) выражения булевой функции n переменных - это дизъюнкция n-терминных (индивидных) конъюнкций, представляющая собой четкую, альтернативную, совокупность (*класс*) индивидов в n-терминном универсуме. Кстати, операция отрицания-дополнения булева выражения равносильна инверсии этой совокупности индивидов. Действительно, из составляющих n-терминный универсум 2^n индивидных конъюнкций (из полной совокупности их) подсовокупность членов СДНФ-выражения соответствует именуемому данным выражением классу, а инверсия этой совокупности, т.е. дополняющая до 2^n подсовокупность, в той же дизъюнктивной форме представляет собой выражение дополнительного класса, является отрицанием-дополнением исходного СДНФ-выражения. Короче говоря, операция отрицания-дополнения СДНФ-выражения сводится к инвертированию совокупности его членов.

Аналогично обнаруживается, что конъюнкция СДНФ-выражений сводится к *пересечению,* а дизъюнкция - к *объединению* совокупностей членов каждого из этих выражений. Например:

$$(xy \vee xy' \vee x'y') \wedge (xy \vee x'y \vee x'y') \equiv xy \vee x'y'$$
$$(xy \vee x'y) \vee (xy' \vee x'y) \equiv xy \vee xy' \vee x'y$$

В совершенной конъюнктивной нормальной форме (СКНФ) выражение представлено конъюнкцией предполных дизъюнкций, т.е. дополнений индивидных конъюнкций. Например, выражение характеристической функции $x \leftrightarrow y$ отношения эквивалентности, состоящее в СДНФ из двух индивидов: $xy \vee x'y'$, в СКНФ оказывается конъюнкцией дизъюнкций: $(x \vee y')(x' \vee y)$, причем дизъюнкции эти суть дополнения индивидов $x'y$ и xy' из СДНФ-выражения функции антиэквивалентности: $xy' \vee x'y$. Операция отрицания-дополнения СКНФ-выражения, как и СДНФ, сводится к инвертированию совокупности его членов. Однако конъюнкции СКНФ-выражений соответствует не пересечение, а *объединение*, и дизъюнкции - не объединение, а *пересечение* совокупности их членов. Примеры:

$$(x' \vee y)(x \vee y') \wedge (x' \vee y) \equiv (x' \vee y)(x \vee y')$$
$$(x' \vee y)(x \vee y') \vee (x' \vee y) \equiv x' \vee y$$
$$\neg(x \vee y) \equiv (x \vee y')(x' \vee y)(x' \vee y')$$

В n-терминном универсуме n-арные элементарные конъюнкции определяют четкие совокупности (*множества*) терминов, тогда как связывание дизъюнкцией порождает нечеткие совокупности (*классы*). И те и другие можно *потерминно* инвертировать, пересекать, объединять, формируя из выражений-операндов выражения-результаты, определяющие искомые совокупности. Примеры:

инверсия: $(x' \vee y)' \equiv x \vee y'$

пересечение: $x' \cap y \equiv (x'y \vee x'y') \cap (xy \vee x'y) \equiv x'y \vee x'y' \equiv x'$

объединение: $x' \cup y \equiv (x'y \vee x'y') \cup (xy \vee x'y) \equiv xy \vee x'y \equiv y$

Несложно выявить алгебраические законы, которым подчинены понимаемые таким образом операции пересечения, объединения и инверсии булевых выражений, т.е. сформулировать положения (аксиомы), формально определяющие эти операции.

Пересечение и объединение, подобно конъюнкции и дизъюнкции, идемпотентны и коммутативны:

$x \cap x \equiv x$ $\qquad$ $x \cup x \equiv x$

$x \cap y \equiv y \cap x$ $\qquad$ $x \cup y \equiv y \cup x$

Они взаимно дистрибутивны одно относительно другого, а также относительно конъюнкции и дизъюнкции:

$(x \cap y) \cup z \equiv (x \cup z) \cap (y \cup z)$ $\qquad$ $(x \cup y) \cap z \equiv (x \cap z) \cup (y \cap z)$

$(x \cap y) \vee z \equiv (x \vee z) \cap (y \vee z)$ $\qquad$ $(x \cup y) \vee z \equiv (x \vee z) \cup (y \vee z)$

$(x \cap y) \wedge z \equiv xz \cap yz$ $\qquad$ $(x \cup y) \wedge z \equiv xz \cup yz$

$(x \wedge y) \cap z \equiv (x \cap z) \wedge (y \cap z)$ $\qquad$ и т.д.

Инверсия булева выражения формально означает инвертирование каждого вхождения каждого из входящих в это выражение терминов с сохранением неизменными всех иных связок. При этом инверсия первичного (элементарного) термина x удовлетворяет аксиомам:

$(x')' \equiv x, \quad x \cap x' \equiv x', \quad x \cup x' \equiv x, \quad x \wedge x' \equiv 0, \quad x \vee x' \equiv 1$

Содержательно инверсия термина означает изменение на противоположный его статуса как члена совокупности. Инвертируются, в сущности, не термины, а их статусы в рассматриваемой совокупности: принадлежность ей термина x заменяется антипринадлежностью (исключенностью принадлежности) - x', антипринадлежность же x' заменяется принадлежностью x. Так что инвертируются не термины, а совокупности. Инверсией же отдельного термина называют, допуская "вольность речи", инверсию однотерминной совокупности, однотерминного булева выражения.

Точно так же, пересечение и объединение - это пересечение и объединение совокупностей. Пересечение формирует совокупность,

содержащую только те термины, которые принадлежат всем пересекающимся совокупностям, т.е. позволяет извлечь из данного набора характеристик общую для них сущность. Объединение, наоборот, признает антипринадлежащими результирующей совокупности лишь те термины, которые антипринадлежат каждой из объединяемых совокупностей, так что формируется наименее строгая характеристика определяемого объекта.

Собственно отдельные термины суть символы элементарных (не конкретизируемых в принятом универсуме) определенностей, конъюнкцией и дизъюнкцией которых, а также их инверсий, образуются простейшие составные выражения булевой алгебры - *элементарные конъюнкции* и *дизъюнкции.* Они представляют собой первичные неэлементарные определенности, к которым также применимы операции конъюнкции и дизъюнкции. Вместе с тем, они интерпретируются и как совокупности элементарных определенностей, преобразуемые посредством их инверсии, пересечения и объединения.

Конъюнкция булевых выражений (КНФ) представляет собой совокупность *необходимых* условий того, что охарактеризовано этой конъюнкцией в целом. Поэтому представленный КНФ-выражением объект имеет место (необходимо дан) лишь в случае, когда даны все члены КНФ. Обратно, из данности КНФ-выражения необходимо следует данность каждого его члена.

Дизъюнкция булевых выражений (ДНФ) определяет характеризуемый ею объект дизъюнктивным (альтернативным) перечнем *достаточных*, но не необходимых, условий - подвыражений более строгих, чем ДНФ-выражение в целом. Это различные конкретизации объектов обозначенного ДНФ-выражением класса. Из данности класса следует *возможность* каждого из относящихся к этому классу объектов (подклассов), становящаяся *необходимостью* при наложении надлежащих дополнительных условий.

Начало иерархически упорядоченной булевой алгебры, или основание ее первой ступени, составляют инверсия и конъюнкция *первичных* (несоставных, недекомпозируемых) терминов. Комбинация ("суперпозиция") этих операций порождает простейший тип составного булева выражения - *элементарную конъюнкцию.* Алгебра элементарных конъюнкций замкнута и функционально полна в том смысле, что инверсии и конъюнкции выражений этого типа суть тоже элементарные конъюнкции, а всякая представимая в виде элементарной конъюнкции функция терминов реализуема посредством операций инверсии и конъюнкции. Порождая элементарные конъюнкции, эти операции применимы затем и к ним самим, наряду с потерминными операциями пересечения и объединения их как совокупностей терминов.

В n-терминном универсуме всего 3^n различных элементарных конъюнкций, из которых 2^n индивидные (без умалчивания терминов), соответствующие четким совокупностям, т.е. множествам присущих характеризуемым ими индивидам первичных определенностей. Неиндивидные ("нечеткие") элементарные конъюнкции именуют неиндивидные классы рассматриваемых "предметов", представляя собой нечеткие совокупности присущих им определенностей. Следует заметить, что понятие индивида ("предмета", "вещи") относительно: n-терминная элементарная конъюнкция представляет собой индивид только в n-терминном универсуме, т.е. при вхождении в нее всех принятых в этом универсуме терминов, а с введением в универсум новых терминов прежняя индивидность n-терминной конъюнкции утрачивается. Таким образом, индивидные конъюнкции отображают сущности "вещей" с предопределенным в универсуме огрублением.

Вторая ступень алгебраической иерархии строится посредством тех же операций инверсии и конъюнкции, но применяемых теперь не к первичным терминам, а к индивидам. Они представлены n-терминными элементарными конъюнкциями и в отличие от терминов, предполагающихся ортогональными, т.е. совмещаемыми друг с другом в любых сочетаниях, попарно несовместимы. Поэтому совокупность индивидов немыслима как нераздельное единство, не может быть "единичной вещью". Она может быть лишь набором, группой взятых вместе, *сосуществующих*, единичных вещей. Так сказать, конъюнкцией их существований.

Существование в универсуме U "вещи", охарактеризованной булевой функцией первичных терминов (атрибутом) f, есть *принадлежность* $f \in U$ - отношение , равносильное присущности U существования f:

$$U = U \wedge Vf$$

Интегральная дизъюнкция ("дизъюнкт") Vf есть дизъюнкция значений, принимаемых атрибутом f на элементах совокупности U, распространенная на всю эту совокупность, т.е. на весь универсум. *Сосуществование* (сопринадлежность совокупности U) "вещей" f и g выражается соответственно конъюнкцией их существований:

$$U = UVfVg$$

Отношения существования и сосуществования выразимы при помощи квантора существования $\exists$ логики предикатов: $Vf = \exists p\, f(p)$, $VfVg = \exists p\, f(p)g(p)$, где p - "предметная переменная". Однако ввиду отмеченной выше относительности понятия "предмета" (индивида) и обусловленной им неоправданной усложненности алгебры, вместо кванторов с предметной переменной целесообразней использовать аналоги интегральной суммы и произведения - интегральные дизъюнкцию и конъюнкцию (дизъюнкт и конъюнкт). При этом алгебра

совокупностей 2-й ступени (совокупностей индивидов) с базисными операциями инверсии и конъюнкции полностью воспроизводит булеву алгебру 1-й ступени, но теперь применительно не к первичным терминам, а к дизъюнктам и конъюнктам.

Конкретная совокупность индивидов в n-терминном универсуме отображается конъюнкцией неинвертированных и инвертированных дизъюнктов от n-арных элементарных конъюнкций. Например, в x,y-универсуме совокупность индивидов xy и $x'y'$, представляющая собой отношение актуальной эквивалентности $x = y$, выражается в виде $\mathrm{V}xy\mathrm{V}'xy'\,\mathrm{V}'x'y\mathrm{V}x'y'$, а нечеткая совокупность индивидов, соответствующая отношению актуального (необходимого) следования $x \Rightarrow y$, получается элиминацией в этом выражении дизъюнкта $\mathrm{V}'x'y$: $\mathrm{V}xy\mathrm{V}'xy'\,\mathrm{V}x'y'$.

В алгебре 1-й ступени эти совокупности отображены выражениями общих атрибутов принадлежащих им индивидов, т.е. дизъюнкциями n-арных конъюнкций (СДНФ-выражениями), представляющими собой отношения потенциальной эквивалентности $x \leftrightarrow y$ в виде $xy \vee x'y'$ и потенциального следования $x \rightarrow y$ в виде $xy \vee \sigma x'y \vee x'y'$. Буква σ означает привходящий статус индивида $x'y$ - не принадлежит с необходимостью и не антипринадлежит. В обычной булевой алгебре подобная возможность отображения привходящего не предусмотрена, поэтому отношение следования представлено в ней "материальной импликацией": $xy \vee x'y \vee x'y'$ - общим атрибутом совокупности $\mathrm{V}xy\mathrm{V}'xy'\,\mathrm{V}x'y\mathrm{V}x'y'$, соответствующей частному случаю отношения следования. Другим частным случаем следования является эквивалентность $\mathrm{V}xy\mathrm{V}'xy'\,\mathrm{V}'x'y\mathrm{V}x'y'$.

Налицо взаимно однозначное соответствие СДНФ-выражения общего для всех элементов совокупности индивидов атрибута дизъюнктному выражению самой этой совокупности: оба выражения содержат одну и ту же информацию, знание одного из них позволяет воссоздать другое. При этом инверсия совокупности индивидов равнозначна булеву отрицанию-дополнению общего атрибута ее членов. Выходит, что в булевой алгебре в качестве базисной операции отрицания избрана инверсия 2-й ступени, тогда как поистине базисная инверсия 1-й ступени (инверсия всех вхождений первичных терминов) просто проигнорирована. Правда, в случае однотерминного выражения эти инверсии неразличимы. Однако, только в нем единственном.

Установленное соответствие СДНФ-выражения общего атрибута индивидов совокупности ее дизъюнктному выражению означает также равнозначность конъюнкции (дизъюнкции) СДНФ-выражений и пересечения (объединения) их дизъюнктных аналогов. Другими словами, конъюнкция (дизъюнкция) булевых выражений

осуществима путем пересечения (объединения) совокупностей членов СДНФ этих выражений, что технически существенно проще традиционной процедуры, использующей законы дистрибутивности.

Истолкование булевой алгебры терминов как теоретико-множественной алгебры индивидов, т.е. реализация булева отрицания инвертированием совокупности членов СДНФ-выражения, а конъюнкции и дизъюнкции путем соответственно пересечения и объединения "СДНФ-совокупностей", радикально упрощает, в частности, процедуру решения булевых уравнений, которая играет в алгебре логики наиважнейшую роль. По мнению самого Буля, предмет логики заключается именно в решении логических уравнений [5], а если иметь в виду математическую логику, то это безоговорочно так, ибо математическая она потому, что сводит умозаключение (вывод, доказательство) к решению уравнения. Впрочем, современная "математическая логика", предпочтя импликацию, уравнениями и вовсе не занимается.

Логическим, или булевым, уравнением называют равенство

$$f = g$$

в котором f, g - булевы функции терминов x_1, x_2, ..., x_n, а знак = символизирует заданность отношения эквивалентности (равносильности), выполняющегося на тех наборах (n-ках) значений терминов x_1, x_2,..., x_n, на которых значения, принимаемые функциями f и g, совпадают. Совокупность таких n-ок алгебраисты называют отношением равенства функций f и g. Запись $f = g$ означает связанность f и g отношением эквивалентности: $(f \leftrightarrow g) = 1$, подобно тому как $x = 1$ означает данность (наличие) термина x.

Решением булева уравнения относительно термина x_i называют рекурсивную функцию $x_i = \varphi(x_1, x_2, x_3, ..., x_n)$, удовлетворяющую заданному уравнению. Но ведь не лишено смысла и решение относительно одного из индивидов, т.е. совокупности терминов, а также относительно дизъюнкции тех или иных индивидов, т.е. некоторой булевой функции терминов, частным случаем которой является и термин x_i.

Положим, что функции f и g (левая и правая части уравнения) представлены СДНФ-выражениями, т.е. дизъюнкцияими индивидов. Ясно, что индивид удовлетворяет уравнению, если он входит в обе его части либо не входит ни в левую, ни в правую часть. Назовем такой индивид *парным* [6, с.47]. Индивиды же, входящие в одну из частей, но не входящие в другую (*непарные* индивиды), уравнению не удовлетворяют. Разделение индивидов на парные и непарные радикально упрощает получение решений, а также тождественные преобразования уравнений. Действительно, выражаемое уравнением отношение сохраняется неизменным при переносе непарных индивидов

из одной части уравнения в другую, а также при добавлении в обе части либо исключении из обеих частей парных индивидов.

Приведение к так называемой "нулевой" форме, в которой правая часть уравнения представляет собой пустую дизъюнкцию (обозначается цифрой "0"), достигается исключением всех парных индивидов и переносом всех непарных в левую часть. "Единичная" форма с обозначаемой цифрой "1" полной дизъюнкцией в правой части получается добавлением в обе части уравнения не представленных в них парных индивидов и переносом из левой части в правую всех непарных, в результате чего в правой части образуется полная дизъюнкция. Примеры:

1) $xyz \vee xyz' = xyz' \vee x'y'z'$ тождественно $xyz \vee x'y'z' = 0$

2) $xy' \vee x'y' = x'y'$ тождественно $xy \vee x'y \vee x'y' =$

$= xy \vee xy' \vee x'y \vee x'y'$, тождественно $xy \vee x'y \vee x'y' = 1$,

тождественно $xy' = 0$

Решение уравнения относительно одного из индивидов, либо термина, либо произвольного атрибута получается путем тождественного преобразования, формирующего в левой части СДНФ-выражение искомого объекта.

Например, решение уравнения из примера 2) относительно термина x получается преобразованием к виду: $xy \vee xy' = xy$, где $xy \vee xy' = x(y \vee y') = x$, т.е. решение в канонической форме будет:

$$x = xy$$

$$x = \begin{cases} 0 \text{ при } y = 0 \\ \text{привходяще при } y = 1 \end{cases}$$

Решение этого же уравнения относительно термина y:

$$xy \vee x'y = xy \vee xy' \vee x'y$$

$$y = xy \vee xy' \vee x'y$$

$$y = x \vee y$$

$$y = \begin{cases} \text{привходяще при } x = 0 \\ 1 \text{ при } x = 1 \end{cases}$$

Решение относительно индивида xy:

$$xy = xy \vee xy'$$

$$xy = x$$

$$xy = \begin{cases} 0 \text{ при } x = 0 \end{cases}$$

$$\lfloor 1 \text{ при } x = 1$$

Решение относительно $x \vee y$:

$$xy \vee xy' \vee x'y = xy \vee xy'$$

$$x \vee y = x$$

$$x \vee y = \begin{cases} 0 \text{ при } x = 0 \\ \lfloor 1 \text{ при } x = 1 \end{cases}$$

Решение уравнения из примера 1) относительно термина x:

$$xyz \vee x'y'z' = 0$$

$$xyz \vee xyz' \vee xy'z \vee xy'z' = xyz' \vee xy'z \vee xy'z' \vee x'y'z'$$

$$x = x(yz' \vee y'z) \vee y'z'$$

$$x = \begin{cases} 1 \text{ при } y = z = 0 \\ \text{привходяще при } y \neq z \\ \lfloor 0 \text{ при } y = z = 1 \end{cases}$$

Как видно, решение уравнения относительно заданного атрибута заключается в преобразовании уравнения к виду, при котором СДНФ-выражение атрибута составляет левую часть в СДНФ-выражении уравнения, обе части которого затем минимизируются. Сущность минимизации СДНФ-выражений в том, что булевы функции терминов, представленные дизъюнкциями индивидных конъюнкций, переводятся обратно в неиндивидное ("несовершенное") представление [3]. Дизъюнкции индивидов порождают термины и неиндивидные классы терминов, подобно тому как конъюнкцией терминов образуются индивиды.

Впрочем, так называемая *двойственность* дизъюнкции и конъюнкции этим не исчерпывается: индивидным конъюнкциям однозначно соответствуют их "дуалы" - предполные дизъюнкции, конъюнкция которых порождает совершенную конъюнктивную нормальную форму (СКНФ) булевых выражений. Возникает точно такая же иерархия ступеней, как рассмотренная выше ассоциирующаяся с СДНФ, однако полностью двойственная, т.е. вместо индивидных конъюнкций теперь предполные n-арные дизъюнкции терминов, дизъюнкция индивидов, составляющая СДНФ, стала конъюнкцией двойственных индивидам предполных дизъюнкций, дизъюнкты индивидов - конъюнктами предполных дизъюнкций.

Формально выражение, двойственное данному, получается взаимозаменой в нем всех конъюнкций дизъюнкциями и всех дизъюнкций конъюнкциями. Содержательно оно есть дополнение инверсии данного, что равносильно инверсии его дополнения:

$$\delta e = \rceil e' = (\rceil e)'$$

Поэтому для преобразования выражения e в двойственную форму надо выполнить над ним (в любом порядке) операции инверсии, дополнения и дуалирования, т.е. взаимозамены конъюнкций с дизъюнкциями. Например, перевод в КНФ выражения $xy \vee x'y \vee x'y'$:

$\neg(xy \vee x'y \vee x'y') = xy'$, $(xy')' = x'y$, $\delta(x'y) = x \vee y'$;

перевод в КНФ выражения $xy \vee \sigma x'y \vee x'y'$:

$\neg(xy \vee \sigma x'y \vee x'y') = xy' \vee \sigma x'y$, $(xy' \vee \sigma x'y)' = x'y \vee \sigma xy'$,

$\delta(x'y \vee \sigma xy') = (x' \vee y)(x \vee y' \vee \sigma)$;

перевод в ДНФ выражения $(x \vee y)(x' \vee y)(x' \vee y')$:

$\neg(x \vee y)(x' \vee y)(x' \vee y') = (x \vee y')$, $\delta(x \vee y') = xy'$, $(xy')' = x'y$.

Проще получается дополнение выражения e, представленное в двойственной форме:

$$\neg e = \delta(e') = (\delta e)'.$$

Это дуал инверсии данного выражения либо инверсия дуала его - процедура, унифицирующая правила де Моргана отрицания конъюнкции $\neg(xy) = \neg x \vee \neg y$ и отрицания дизъюнкции $\neg(x \vee y) = \neg x \neg y$.

Существенно, что КНФ есть совокупность *необходимых*, а ДНФ - *достаточных* условий того, что представлено выражением в целом. Двойственность формы сопряжена с фундаментальной парой содержательных критериев - необходимости и достаточности.

Произвольное выражение булевой алгебры, как в ДНФ, так и в КНФ, представимо при помощи трех функторов-связок: инверсии, конъюнкции и дизъюнкции, составляющих базис этой алгебры. Прочие функторы, такие как дополнение, дуалирование, пересечение объединение, подчинение, обозначают операции над выражениями, в нормальных формах не используемые ("небазисные"). Дизъюнкты и конъюнкты, а также символ привходящего σ, к "классической" булевой алгебре не относятся и составляют необходимое расширение, вернее, развитие ее, связанное с построением 2-й ступени и с допущением нечетких атрибутов совокупности индивидов.

Дизъюнктивная форма начинается с элементарной конъюнкции, выражающей в ее n-арном варианте *единичное* (индивид) в n-терминном универсуме. Все прочие, нечеткие (неиндивидные) выражения в ДНФ суть дизъюнкции элементарных конъюнкций. В частности, дизъюнкция 2^n-1 индивидов есть "предполная дизъюнкция", составляющая дополнение недостающего в ней 2^n-го индивида, в совокупности с которым она становится "полной", т.е. универсумом.

Конъюнктивная форма устроена аналогично, но начинается с противоположного - с элементарной дизъюнкции, которая в n-арном варианте является предполной дизъюнкцией, т.е. дополнением противоположного ей индивида до универсума. Совершенной

нормальной формой предполной дизъюнкции называется дизъюнкция 2^n-1 индивидов, а минимальной формой - n-арная элементарная дизъюнкция.

Литература

1. Брусенцов Н.П. Искусство достоверного рассуждения. Неформальная реконструкция аристотелевой силлогистики и булевой математики мысли. - М.: Фонд "Новое тысячелетие", 1998.

2. Брусенцов Н.П., Деркач А.Ю. Трехзначная логика, нечеткие множества и теория вероятности. // Программные системы и инструменты. Тематический сборник №2. Под редакцией чл.-корр. РАН Л.Н. Королева. - М.: Факультет ВМиК МГУ, 2001. С. 88-91.

3. Брусенцов Н.П., Владимирова Ю.С. Трехзначный минимизатор булевых выражений. // Программные системы и инструменты. Тематический сборник №2. Под редакцией чл.-корр. РАН Л.Н. Королева. - М.: Факультет ВМиК МГУ, 2001. С. 205-207.

4. Аристотель. Топика. // Сочинения в четырех томах. Том 2. - М.: "Мысль", 1978.

5. Брусенцов Н.П., Владимирова Ю.С. Решение булевых уравнений. // Методы математического моделирования. - М.: "Диалог МГУ", 1998. С. 59-68.

6. Брусенцов Н.П. Начала информатики. - М.: Фонд "Новое тысячелетие", 1994.

Леонов М.В., Мошкин К.Б., Леонов В.М.

XML как средство модернизации унаследованных баз данных*

Введение

В статье предложен подход к решению проблемы унаследованных баз данных и информационных систем с использованием языка XML. Представлена разработанная на основе такого подхода система, являющаяся модернизированным вариантом ИПС GNOM [1] с базой данных формата dbf, разработанной около 15 лет назад для хранения таксономических данных по родам семейства Зонтичных в сотрудничестве с проф. М.Г.Пименовым из Ботанического сада МГУ [3].

Актуальность задачи

Проблема унаследованных баз данных заключается в следующем. Довольно часто данные, накопленные в процессе эксплуатации давно разработанных ИПС, продолжают сохранять свою актуальность и востребованность. Развитие же таких систем, с морально устаревшей оболочкой и платформой, также как и эксплуатация, порождает иногда многочисленные технические проблемы. Простейший пример – чтобы получить результаты запросов, оформленные в стиле Word for Windows в системе, разработанной для DOS, приходится прибегать к ухищрениям, вроде вставок в результат запроса специальных символов, впоследствии обрабатываемых макросами программы Word. Иногда в процессе эксплуатации у пользователей возникает потребность в запросах к БД, реализовать которые не всегда просто из-за того, что они не рассматривались при первоначальном проектировании. Кроме того, существуют естественные трудности при организации доступа к таким БД через Интернет.

* Работа выполнена в соответствии с задачами проекта, поддержанного грантом **РФФИ 02-07-90303**

Суть подхода и выбор инструментария

Из возможных способов решения данной задачи был выбран следующий, основанный на применении XML-технологии. На сегодняшний день нам неизвестно применение этой технологии в таксономической ботанике, хотя именно эта область богата слабоструктурированными данными, для которых в XML считается весьма подходящим средством. К тому же XML обоснованно претендует на глобальный кроссплатформенный способ управления, приема и передачи данных. Можно сказать, что XML дает новую возможность представления данных, в некотором смысле промежуточную между текстовым представлением и представлением в традиционных БД. Относительная простота XML-документов дает возможность обсуждения проектов хранения данных в таком виде с «компьютерно-продвинутыми» пользователями-ботаниками, для которых HTML уже стал привычным инструментом представления данных. Это также немаловажное преимущество.

Итак, наш подход заключается в создании XML-документов, содержащих данные унаследованной БД, связанных между собой в соответствии с моделью предметной области. Следуя предложению проф. С.Д.Кузнецова, мы назвали совокупность этих файлов XML-образом фрагмента предметной области, чтобы избежать возможной дискуссии, если бы для этого употребили словосочетания «XML-модели» или «XML-БД», хотя и то, и другое также отражают суть конструкции.

Для реализации запросов был выбран популярный, имеющий версии и для UNIX и для Windows язык PHP4, в котором уже предусмотрены модули для работы с XML-документами, а также web-сервер Apache. Визуализация результатов запросов осуществляется с помощью браузера Internet Explorer 5 с заранее подготовленными XSL-таблицами.

Заметим, что описываемый подход можно обобщить на случай нескольких БД для одной предметной области, что иллюстрирует рис.1.

Этапы решения задачи

На первом этапе была разработана структура XML-документов фрагмента предметной области, соответствующая объектам таксономической ботаники в пределах, необходимых для представления номенклатора родов семейства Зонтичных [2]. (Номенклатор родов – это справочник, содержащий важнейшие данные по синонимии

названий, первоописанию, ссылки на литературные источники, типовые образцы, географическое распространение по континентам и их регионам, число видов и др.). Другими словами, разработана система тэгов, адекватно описывающая предметную область, учитывающая возможные запросы, а также структура документов, по возможности обеспечивающая неизбыточность данных.

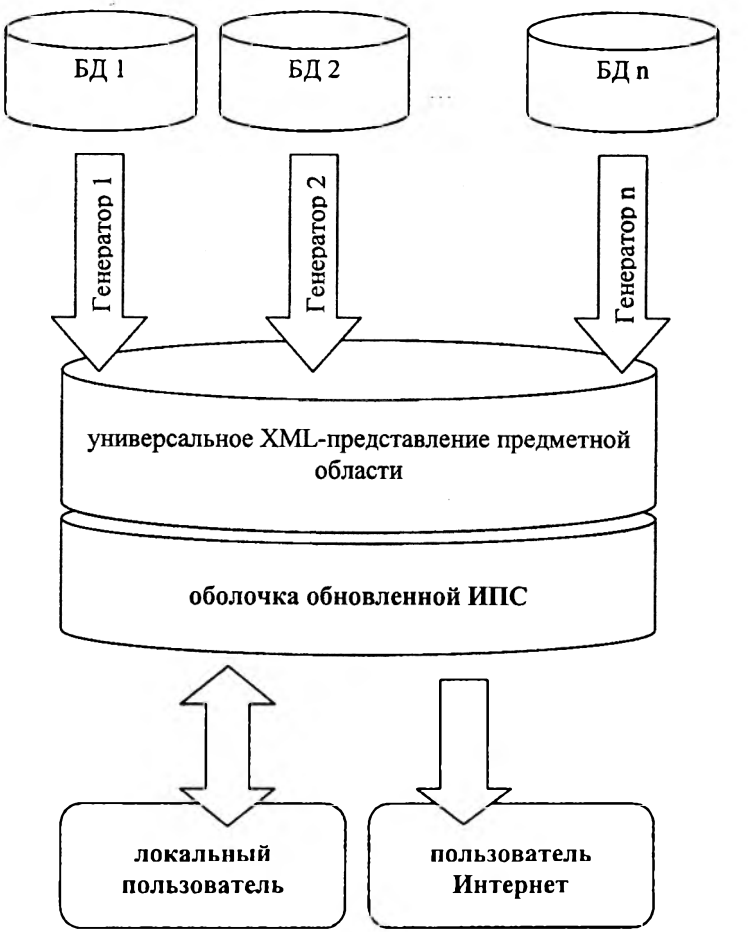


Рис.1. Предлагаемое решение проблемы унаследованных БД

На втором этапе был спроектирован и реализован программный преобразователь файлов формата dbf нашей БД GNOM в набор файлов - XML-документов, связанных между собой ссылками. Эта программа

реализована в системе Delphi, ее интерфейс позволяет изменять названия тэгов в генерируемых документах, а также редактировать исходные файлы.

Заметим, что несмотря на общеизвестный факт о «многословности» XML-представления данных, объем в байтах получившегося XML-образа примерно равен объему совокупности файлов исходной БД. Дело здесь, по-видимому в том, что реляционная БД также не самым эффективным (в смысле затрат памяти) образом хранит данные такой структуры, как наша. Правда, названия тэгов нами были выбраны хотя и мнемоничными, но по возможности краткими.

Третий этап заключался в создании пользовательской оболочки для запросов к XML-номенклатору, в программировании библиотеки скриптов, реализующих эти запросы и создании XSL-таблиц для визуализации выводимых данных. При написании скриптов на PHP использовался стандарт Xpath – стандарт адресации элементов структурированных деревообразных документов. Этот стандарт существенно облегчает адресацию нужных элементов, но не позволяет осуществлять поиск. Поэтому в сочетании с Xpath для синтаксического разбора конструкций и поиска в XML-документах приходится использовать модуль domxml.

Окончательный проект оболочки предполагает реализацию всех тех запросов, которые присутствовали в системе GNOM – получение краткого и полного списков родов, литературы, запросов по географии, вышестоящим таксонам и др., а также тех которых не было, но необходимость в которых есть у коллег-ботаников, в частности, поиск в по списку типовых видов. Кроме того, оболочка дает возможность вносить изменения в XML-документы в терминах предметной области (то есть добавлять, исключать или редактировать данные по роду, литературному источнику, географическому региону) Естественно, интерфейс у новой системы несравненно современнее, чем у ее DOS-прототипа [1].

Заключение

Практический результат реализации изложенного статье подхода к решению проблемы унаследованных баз данных – новый вариант ИПС по родам семейства Зонтичных с современным интерфейсом – подтверждает целесообразность этого способа по крайней мере для баз данных объемом в несколько тысяч записей. Кроме простоты, он имеет ряд несомненных достоинств. Во-первых, обновленная таким образом система может использоваться не только на локальной машине, но и через Интернет. Во-вторых, благодаря прозрачности и простоте структуры XML-документов пользователь

может быть вовлечен в процесс улучшения системы почти на равных с программистом. В-третьих, достаточно просто решается ряд технических проблем, существующих в DOS-системах (например, хранение и воспроизведение текстов с символами национальных алфавитов типа умляутов и т.д.). В–четвертых, вследствие наблюдаемого бурного развития средств обработки XML-документов можно ожидать появления более эффективных средств обработки XML-документов, которыми можно воспользоваться при развитии системы для улучшения ее характеристик в будущем.

Литература

1. Пименов М.Г., Леонов М.В. Компьютерная база данных по номенклатуре родов Umbelliferae мира. 1992, Бот. Журнал, 77, 12, с.69-77.

2. M.G.Pimenov, M.V.Leonov. The genera of the Umbelliferae. A nomenclator. 1993. Royal Botanic Gardens, 156pp. Kew.

3. Леонов М.В. Комплекс программ для автоматизации исследований в таксономической ботанике. // Программные системы и инструменты. Тематический сборник № 2, М.: Изд-во факультета ВМиК МГУ, 2001, стр. 168-172.

Петровский М.И.

Мера сходства для сравнения прецедентов в системах анализа данных, поддерживающих стандарт OLEDB for DM

Введение

Система интеллектуального анализа данных (Data Mining) - класс программных систем поддержки принятия решений, целью которых является выявление закономерностей в хранилищах данных. Существует несколько типовых задач классификационного анализа, для решения которых используются системы анализа данных [2,6]. Собственно задача классификации - отнесение объектов базы данных к заранее определенным категориям. Фактически задача классификации — это классическая задача распознавания, где по обучающей выборке система относит новый объект к той или иной категории. Вторая задача - прогнозирование - состоит в том, чтобы предсказать по значениям одних полей объекта значения остальных. Третья задача - кластеризация, то есть выделение компактных подгрупп объектов с близкими свойствами. Она, как правило, предшествует задаче классификации, поскольку позволяет определить группы объектов. Четвертая задача - нахождение исключений, то есть поиск объектов, которые своими характеристиками сильно выделяются из общей массы. Системы анализа данных используют методы и математический аппарат из различных областей, таких как прикладная статистика, искусственный интеллект, нейронные сети, информационный поиск, теория нечетких множеств и другие. В рамках каждого из этих направлений разработано много методов решения задач классификационного анализа, но независимо от математического аппарата, методы, используемые в системах анализа данных имеют ряд особенностей, которые должны учитываться. Это, во-первых, разнородность данных, т.е. наличие как числовых, так и номинальных (дискретных) атрибутов у классифицируемых объектов. Во-вторых, частичная структурированность данных, т.е. объект классификации может быть срезом n-мерного куба или элементом реляционного отношения, в зависимости от типа источников данных. В-третьих, количество объектов и их атрибутов может быть достаточно большим,

что делает практически невозможным применение алгоритмов с экспоненциальной сложностью. И наконец, в-четвертых, это отсутствие априорной информации и каких-либо гипотез, т.е. система должна все делать «с нуля». Большинство классических алгоритмов классификации, выявления исключений и кластеризации ориентированы на работу с данными простой структуры, когда объект классификации представляет собой вектор значений фиксированной длины. Но реальные задачи, решаемые с помощью систем анализа данных, приводят к более сложным структурированным данным. В стандарте OLEDB for DM [1] предлагается формат для описания таких структур, но интерпретация этого описания ложится на алгоритм анализа данных. В связи с этим становится актуальной проблема определения меры сходства или расстояния на множестве таких объектов, это позволит использовать в системах анализа данных, поддерживающих стандарт OLEDB for DM, многие классические алгоритмы классификации, кластеризации и выявления исключений. В данной работе рассматривается подход к определению мера сходства с помощью метода потенциальных функций [3,4].

Спецификация OLEDB for Data Mining

Для успешного использования системы анализа данных она должна быть легко интегрируемы в пользовательские приложения и уметь работать с СУБД и хранилищами данных различных производителей. Для этого она должна соответствовать принятым промышленным стандартам. Нами рассматривается стандарт OLE DB for DM, как один из наиболее успешных и популярных. Он специфицирует интерфейс взаимодействия между системой анализа данных и клиентским приложением, кроме того, он позволяет использовать различные источники данных, поддерживающих стандарт OLEDB, в том числе и многомерные. Спецификация включает описание COM интерфейсов, являющихся расширением стандарта OLEDB, которые должна реализовывать система анализа данных, SQL-подобный язык взаимодействия с клиентским приложением, а также поддерживает стандарт PMML для экспорта и импорта описания моделей и взаимодействия с другими системами анализа данных. Ключевым понятием в спецификации OLE DB for DM является понятие модели анализа данных. Модель анализа данных может рассматриваться как «виртуальная» таблица специального вида, хранящая обнаруженные закономерности (например, прототипы кластеров, узлы деревьев решений, параметры распределений, весовые коэффициенты и топологию нейронной сети и т.п.). Создание модели осуществляется с помощью запроса CREATE MODEL, аналогичного запросу CREATE

TABLE в SQL. Данный запрос, помимо названия модели, задаёт структуру классифицируемого объекта, т.н. прецедента. Формально структура прецедента элемент реляционного отношения, т.е. так же задается список и типы атрибутов, но обязателен первичный ключ идентифицирующий прецедент, кроме того, есть возможность использовать вложенные отношения в качестве атрибутов. Обучение модели происходит с помощью запроса INSERT INTO, при этом на вход модели подается тренировочный набор, представляющий собой коллекцию прецедентов сформированную как правило выборкой из внешнего источника данных. Анализ данных с помощью обученной модели происходит с использование запроса SELECT PREDICTION JOIN и специальных функций типа Cluster(), PredictProbability() и др., при этом к коллекции анализируемых прецедентов, сформированной также выборкой из внешнего источника данных, применяется обученная модель.

Структура прецедента в OLEDB for DM

Как говорилось выше классифицируемым объектом в нашем случае является прецедент (OLEDB for DM case), т.е. элемент (картеж) реляционного отношения, имеющий как числовые, так и номинальные (дискретные) атрибуты, а также вложенные реляционные отношения.

Case ID	A_1	A_2	...	T_1			T_2		
...	...	...	...	T_1 key	T_1 A_1	...	T_2 key	T_2 A_2	...
				...	...	...	...	...	...
				...	...	...	...	...	...
				...	...	...			
				...	...	...			

Пример прецедента с тремя нетабличными и двумя табличными атрибутами

Обычный подход к определению метрики для такого рода объектов, применяемый, в частности в продуктах Microsoft, это «развертывание» прецедента в вектор номинальных и числовых атрибутов, т.е. вместо каждого значения атрибута из вложенной таблицы добавляется новый «основной» атрибут с именем,

сформированным из имени таблицы, имени вложенного атрибута и значения первичного ключа во вложенной таблице. Т.е. например для случая изображенного на рисунке, вместо значения вложенного атрибута T_1A_1 в первой строке будет добавлен четвертый нетабличный атрибут с именем $T_1A_1(1)$ и т.д. После чего, в зависимости от используемого алгоритма анализа данных, применяется либо дискретизация числовых значений, либо определение вероятностной метрики, делая предположения о распределениях значений атрибутов, либо используется комбинации евклидового расстояния для числовых атрибутов и например коэффициента Жаккара [7] для номинальных и т.п. Но эти подходы имеют два общих недостатка. Во-первых, в результате получается сильно разряженный вектор очень большой размерности, например, если во вложенной таблице тысяча записей, то размерность вектора увеличится на тысячу, даже если реально, в каждом из прецедентов использовалось не более десяти значений из этой вложенной таблицы. Во-вторых, основные атрибуты и атрибуты вложенных таблиц получаются равноправными, т.е. теряется структура прецедента. Чтобы избежать этих недостатков мы использовали метод потенциальных функций (kernel-functions) [3] для определения метрики в пространстве классифицируемых объектов - прецедентов.

Метод потенциальных функций для определения меры сходства

Кратко, идея метода потенциальных функций (kernel-функций) [3] состоит в следующем. Исходное пространство объектов X с помощью нелинейного отображения ф отображается в пространство большей или бесконечной размерности H, называемое пространством характеристик, причем скалярное произведение в этом пространстве выражается через потенциальную функцию K. Кроме того, явный вид отображения ф не важен и как правило неизвестен, оно неявно определяется видом функции K, а для вычисления расстояния можно, например, воспользоваться формулой:

$$d(x,x') = \sqrt{K(x,x) - 2K(x,x') + K(x',x')}$$

Хотя существуют и другие определения расстояний через потенциальные функции [3,4]. Для простоты можно рассматривать K как меру сходства, т.е. вещественную симметричную, положительно определенную функцию:

$K : X \times X \to R$

$\forall x, x' \in X \; K(x, x') = K(x', x)$

$\forall N \geq 1 \; u \; \forall x_1, x_2, ..., x_N \in X \; u \; \forall c_1, c_2, ..., c_N \in R \; u \; K_{ij} = K(x_i, x_j)$

$\sum_{i,j} c_i c_j K_{ij} \geq 0$

Меры сходства, определенные с помощью потенциальных функций изначально применялись в основном в задачах распознавания образов и теории статистических решений. В частности, в методах минимизация эмпирического риска, максимальной границы и т.д., причем исходным пространством было как правило вещественное пространство, т.е. классифицируемым объектом был вещественный вектор. Но в последнее время появились работы, предлагающие методы определения kernel-функций для дискретных структурированных объектов, таких как строки, деревья, графы, эти методы с успехом применялись и применяются в прикладных задачах классификации текстовых документов, анализа цепочек протеинов и др [3,4]. В работе [4] предлагается общий подход для определения kernel-функции на множестве объектов сложной дискретной структуры, если для составных частей такого объекта kernel-функции уже определены. В этой работе предложена теорема о замыкании, доказывающая факт того, что получающаяся в результате его метода комбинация kernel-функций так же является kernel-функцией. Рассмотрим этот метод более подробнее.

Пусть $x \in X$ некоторая составная структура данных, состоящая из D «частей» $x_1, ..., x_D$, где $x_d \in X_d$ и на $X_d \times X_d$ определена kernel-функция K_d. Отношение R на $X_1 \times ... \times X_D \times X$ выражает тот факт, что для заданного x, $\vec{x} = x_1, ..., x_D$ есть его составная часть и в этом случае существует $R(\vec{x}, x)$. Тогда $R^{-1}(x) = \{\vec{x} : R(\vec{x}, x)\}$. Для конечных R в [4] доказывается, что можно определить K на $X \times X$:

$$K(x, y) = \sum_{\vec{x} \in R^{-1}(x), \vec{y} \in R^{-1}(y)} \prod_{d=1}^{D} K_d(x_d, y_d) \quad (*)$$

Реляционная алгебра с вложенными отношениями

Для того чтобы формально определить меру сходства для прецедентов, являющихся элементами реляционных отношений с вложенными таблицами мы будем использовать определения

реляционной алгебры с вложенными отношениями, предложенные в работе [5]. Кратко обозначим основные моменты.

$U = \{A_1,..., A_p\}$ универсальное множество атрибутов.

Деревом схемы T, заданным на множестве атрибутов $W \subseteq U$, будем называть корневое дерево такое что с каждым из его узлов связано подмножество W атрибутов, попарно непересекающихся.

Над T определены следующие операции:

Att(n) –возвращает множество атрибутов, связанных с узлом n

Root(T) – возвращает корневой узел T

S(T) –объединение Att(n) по всем узлам n

Реляционная схема с вложенными отношениями R(T), заданная на множестве атрибутов U деревом T определяется рекурсивно:

1. Если T пусто, то R(T) тоже пусто
2. Если в T только один узел n и V=Att(n), то R(T)=V
3. Если V=Att(Root(T)) и $T_1,...,T_s$ поддеревья первого уровня в T, то

$$R(T) = V \cup \{(R(T_1)^*),...,(R(T_s)^*)\}$$

$Z(R(T)) = R(T) \cap U$ - множество атомарных, нетабличных атрибутов

$H(R(T)) = R(T) - Z(R(T))$ - множество табличных атрибутов

$D = \{D_1,..., D_p\}$ - домены на которых определены атомарные атрибуты, т.е. $D_i = DOM(A_i)$. Мы будем рассматривать атомарные атрибуты двух типов – числовые, т.е. те для которых определены расстояние и порядок, и номинальные (дискретные).

Домен для схемы R(T) тоже определяется рекурсивно

1. Если T пусто, то DOM(R(T)) тоже пусто
2. Если в T только один узел n и V=Att(n) и $V = \{A_1,..., A_m\}$, то $DOM(R(T)) = DOM(V) = DOM(A_1) \times ... \times DOM(A_m)$
3. Если V=Att(Root(T)) и $T_1,...,T_s$ поддеревья первого уровня в T, то

$DOM(T_1,...,T_s) = P(DOM(R(T_1))) \times ... \times P(DOM(R(T_s)))$, где P(X) есть множество всех подмножеств X. Получаем $DOM(R(T)) = DOM(V) \times DOM(T_1,...,T_s)$

Отношением r* на R(T) есть элемент P(DOM(R(T)). Т.е. r* конечное множество картежей на R(T).

Определение меры сходства прецедентов в OLEDB for DM

Определим теперь kernel-функцию для прецедентов OLEDB for DM. Пусть x и y прецеденты, а структура прецедентов задана с помощью реляционной схемы с вложенными отношениями R(T), тогда потенциальная функция K должна быть определена на множестве $DOM(R(T))\times DOM(R(T))$. Определим ее рекурсивно. Рассмотрим сначала случай, когда R(T) не имеет вложенных отношений.

1. R(T) не имеет вложенных отношений

В этом случае в T только один узел n, V=Att(n) и $V=\{A_1,...,A_m\}$

Тогда $x, y \in DOM(V) = DOM(A_1)\times...\times DOM(A_m)$

и используя (*) можно определить kernel-функцию как

$$K_{DOM(V)}(x,y)=\prod_{j=1}^{m} K_j(x_j,y_j)\ \text{, где}\ x_j, y_j \in DOM(A_j)$$

Здесь возможны два варианта, либо A_j – номинальный атрибут либо числовой.

1.1. A_j - числовой атрибут

В этом случае будем использовать стандартную экспоненциальную потенциальную функцию, так называемую RBF-kernel с параметром q_j и нормированными x_j и y_j:

$$K_j(x_j,y_j)=\exp\left[-q_j\frac{\|x_j-y_j\|^2}{\sigma_j{}^2}\right]$$

1.2. A_j - дискретный атрибут

Пусть A_j принимает n различных значений, для каждого из них известна вероятность (т.е. частота данного значения для данного атрибута в тренировочном наборе). Будем обозначать $p(x_j)$ вероятность того, что значение атрибута A_j есть x_j. Будем рассматривать значение дискретного атрибута как вектор длины n, где i-

ая компонента вектора равна 1, если значение атрибута равно i-ому дискретному значению из $DOM(A_j)$, иначе 0. Т.е.:

$$x_j^i = \begin{cases} 1, x_j = i \\ 0, x_j \neq i \end{cases}$$

Теперь так же как и в случае с числовым атрибутом, используя стандартную экспоненциальную потенциальную функцию, параметром q_j, получаем

$$K_j(x_j, y_j) = \prod_{i=1}^{n} \exp\left[-q_j \frac{\left\| x_j^i - y_j^i \right\|^2}{(\sigma_j^i)^2} \right]$$

σ_j^i-дисперсия i-го столбца

Очевидно, что если $x_j = y_j$, то $K_j(x_j, y_j) = 1$, рассмотрим случай когда $x_j \neq y_j, x_j = i_x, y_j = i_y$. Тогда:

$$K_j(x_j, y_j) = \exp\left[-q_j \frac{1}{(\sigma_j^{ix})^2} \right] \exp\left[-q_j \frac{1}{(\sigma_j^{iy})^2} \right]$$

$\sigma_j^{ix} = p(x_j)(1 - p(x_j))$

Для дискретного атрибута в случае $x_j \neq y_j$ имеем:

$$K_j(x_j, y_j) = \exp\left[-q_j \left(\frac{1}{(p(x_j)(1 - p(x_j)))^2} + \frac{1}{(p(y_j)(1 - p(y_j)))^2} \right) \right]$$

И в результате для прецедентов x и y в случае, если R(T) не имеет вложенных отношений получаем:

$$K_{DOM(V)}(x, y) = \prod_{i \in Num} e^{-q_i \frac{(x_i - y_i)^2}{\sigma_i^2}} \prod_{j \in Discr \wedge x_j \neq y_j} e^{-q_j \left(\frac{1}{(p(x_j)(1 - p(x_j)))^2} + \frac{1}{(p(y_j)(1 - p(y_j)))^2} \right)}$$

Num – множество индексов, соответствующих числовым атрибутам
Discr – множество индексов, соответствующих дискретным атрибутам

2. R(T) имеет вложенные отношения

ПустьV=Att(Root(T)), а $T_1,...,T_s$ поддеревья первого уровня в T, тогда $x, y \in DOM(R(T)) = DOM(V) \times DOM(T_1,...,T_s)$ и по (*) получаем

$$K(x,y) = K_{DOM(V)}(x_V, y_V) \prod_{t=1}^{s} K_t(x_t, y_t), \text{ где } x_t, y_t \subseteq P(DOM(T_t))$$

Пусть $Keys_t$ – множество первичных ключей табличного атрибута t, соответствующего поддереву T_t и мы можем рассматривать k-й картеж в x_t как $x_t(k) \in Dom(T_t)$, где $k \in Keys_t$. Тогда мы можем определить

$$R^{-1}(x_t) = \{x_t(k) \mid k \in Keys_t\} \text{ и } R^{-1}(y_t) = \{y_t(k) \mid k \in Keys_t\}$$

Отсюда, используя (*) и добавляя условие, что $K_t(x_t(k_x), y_t(k_y)) = 0$, если $k_y \neq k_x$ получаем:

$$K_t(x_t, y_t) = \sum_{\vec{x} \in R^{-1}(x_t), \vec{y} \in R^{-1}(y_t)} K(\vec{x}, \vec{y}) = \sum_{k \in Keys(t)} K(x_t(k), y_t(k)).$$

В результате окончательная формула для вычисления степени сходства для прецедентов x и y:

$$K(x,y) = \prod_{i \in Num} e^{-q_i \frac{(x_i - y_i)^2}{\sigma_i^2}} \prod_{j \in Discr \wedge x_j \neq y_j} e^{-q_j \left(\frac{1}{(P(x_j)(1-P(x_j)))^2} + \frac{1}{(P(y_j)(1-P(y_j)))^2} \right)} \times$$

$$\times \prod_{t=1}^{s} \sum_{k \in Keys_t} K(x_t(k), y_t(k))$$

q_i – параметр kernel-функции, который может интерпретироваться как коэффициент «важности» i–го нетабличного атрибута, т.е. чем больше q_i, тем больше «вклад» i–го атрибута в общее значение потенциальной функции.

Заключение

В работе рассмотрена проблема определения меры сходства для сравнения прецедентов в системах анализа данных, работающих с разнородной структурированной информацией, и удовлетворяющих спецификации OLEDB for DM. Предложен подход к определению меры сходства, использующий метод потенциальных функций. Формально показано, что для реляционной модели с вложенными отношениями данная мера сходства является потенциальной функцией. Это позволяет

использовать в системах анализа данных, удовлетворяющих спецификации OLEDB for DM, широкий класс алгоритмов классификации, кластеризации и выявления исключений, работающих с мерой сходства на основе потенциальной функции, т.н. kernel-based и SVM алгоритмы [3], а также ряд классических алгоритмов, использующих понятие расстояния на множестве классифицируемых объектов.

Литература

1. OLE DB for Data Mining Specification. Version 1.0. Microsoft Corporation, 2000. < http://www.microsoft.com/data/oledb/dm.htm>
2. Jiawei Han и Micheline Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, 2000
3. Bernhard Scholkopf, Alexander J. Smola. Learning with Kernels. The MIT Press Cambridge, London England, 2002
4. Mark Levene, George Loizou, A Fully Precise Null Extended Nested Relational Algebra, Fundamenta Informaticae, Vol. 19, pp. 303-343, 1993.
5. D. Haussler, Convolution Kernels on Discrete Structures, Technical Report CSD-TR-98-11 from Royal Holloway, University of London, 15 January 1999
6. M. Chen, J. Han, and P. S. Yu. Data mining: an overview from a database perspective. IEEE Trans. On Knowledge And Data Engineering, 8:866–883, December 1996.

Раздел II
Имитационное моделирование и синтез систем передачи информации

Машечкин И. В., Веселов Н. А.

Оптимизация пропускной способности сетей, построенных по схеме Ч. Клоза, в условиях квазистатического размещения абонентов и отсутствия информации о загруженности системы

Одной из задач, стоящих перед алгоритмами маршрутизации и управления соединениями является оптимизация пропускной способности сети передачи данных. Это имеет большое значение для эффективного и быстрого обмена информацией, а следовательно, для экономичной работы всего вычислительного комплекса, составной частью которого является рассматриваемая сеть. Всвязи с развитием вычислительной техники, появляются новые методы и технологии передачи данных и новые подходы к постоению систем и сетей связи, что влечёт за собой появление новых задач, связанных с разработкой и исследованием методов оптимизации пропускной способности сетей.

В настоящей работе пойдёт речь об оптимизации пропускной способности сетей связи, построенных с использованием топологии, предложенной Ч. Клозом в 1956 году [3]. Изначально рассматриваемая схема соединения предназначалась для построения сетей с коммутацией каналов, таких, например, как коммутаторы телефонных станций, и позволяла сократить сложность коммутационной схемы (количество полюсов коммутации), сохранив при этом пропускную способность такую же или близкую к пропускной способности полносвязной коммутационной схемы. При этом, рассматривались модели ординарной коммутации, когда любое соединение может быть установлено только между свободной парой вход - выход схемы и некоторые случаи неординарной коммутации, когда один вход схемы может связываться с несколькими свободными выходами одновременно. Для этих моделей комбинаторные и вероятностные свойства рассматриваемой топологии хорошо изучены. Однако, все известные на данный момент результаты применимы только для коммутационных схем, имеющих общее управляющее устройство, и для схем, в которых информация о возникновении блокировки соединения

становится известна и может быть обработана непосредственно после её возникновения.

Нами будет рассмотрена модель сети связи с коммутацией каналов, в которой отсутствует любая форма централизованного управления соединениями, а абоненты функционируют независимо и принимают решения о маршрутах установления соединений самостоятельно. Кроме того, протокол обработки состояния блокировки вызова реализован аппаратно, таким образом, что информация о её возникновении не может быть получена и использована непосредственно сразу после этого события. Сеть строится по схеме Клоза, а абоненты сети располагаются на входах и выходах этой схемы. Такой выбор обусловлен тем, подобные системы существуют и широко применяются на практике для построения сетей связи высокопроизводительных вычислительных комплексов. Например, аналогичные особенности функционирования можно выделить в коммуникационной среде Myrinet, созданной и поддерживаемой фирмой Myricom, Inc. [1], [2]. Схема соединения Ч. Клоза также выбрана неслучайно, так как рассматриваемая топология часто используется при построении коммутаторов [2], в качестве примера можно указать на Myrinet Switch 2000, также разработанный фирмой Myricom, Inc. [1].

В связи с тем, что известные методы не могут обеспечить эффективное управление соединениями в данной ситуации, в реальных сетях работают статические и неадаптивные алгоритмы, что приводит к неэффективному использованию пропускной способности сети, а следовательно и к неэкономичному функционированию системы в целом.

Далее будет получено достаточное условие для множества абонентов сети, которое гарантирует существование расположения соединений внутри схемы, обеспечивающего бесконфликтное взаимодействие абонентов. Данное условие является обобщением известной теоремы Слепяна - Дьюгида [4]. Также будет рассмотрен новый подход к управлению соединениями, использующий теорию хроматических графов и предложена эвристика, на основе методов приближённой раскраски графов. В конце статьи мы опишем один из возможных сценариев применения вышеозначенного подхода в реальной системе.

Описание математической модели

Остановимся более подробно на описании схемы работы узлов и построим математическую модель функционирования сети в целом.

Сеть связи задаётся графом $G=(U,V)$, где U - вершины (узлы), V - ребра (линии связи). Каждая вершина графа принадлежит к одному из двух типов: узлы с буферной памятью для хранения пакетов (оконечные устройства или хосты) и узлы, не имеющие таковой (коммутаторы).

У каждого коммутатора для организации взаимодействия с другими коммутаторами или оконечными устройствами имеется множество входных портов и множество выходных портов. Передача данных осуществляется при помощи однонаправленных линий связи, каждая из которых соединяет некоторый вход с некоторым выходом. Рассматриваются полносвязные коммутаторы, то есть способные осуществлять соединение любого свободного входа с любым незанятым выходом. На рис. 1 дано схематическое изображение коммутатора с n входными портами и m выходными портами.

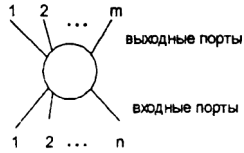


Рис 1. Схематическое изображение простого коммутатора с n входными и m выходными портами.

Все оконечные устройства, в свою очередь, делятся на две группы: узлы, которые могут отправлять сообщения, и узлы, способные эти сообщения принимать. Одни располагаются на входах коммутационной схемы, другие, соответственно, на её выходах. Мы не будем вводить различные символы для обозначения тех и других, так как из контекста всегда будет ясно о каком типе идёт речь. Взаимодействие хостов с сетью осуществляются через единственный входной порт (или выходной порт соответственно), работающий аналогично портам коммутатора. Особенностью оконечных устройств является то, что только они могут служить источниками и приёмниками сообщений, пересылаемых по сети. На рис. 2 дано схематическое изображение оконечного устройства.

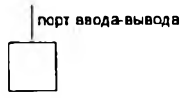


Рис 2. Схематическое изображение оконечного устройства.

Сеть связи строится путём соединения простых коммутаторов по трёхкаскадной схеме Клоза с разделяемыми входами и выходами, что соответствует классическому определению, данному в [3]. К свободным портам схемы подключаются оконечные устройства, которые и являются взаимодействующими абонентами в данной сети.

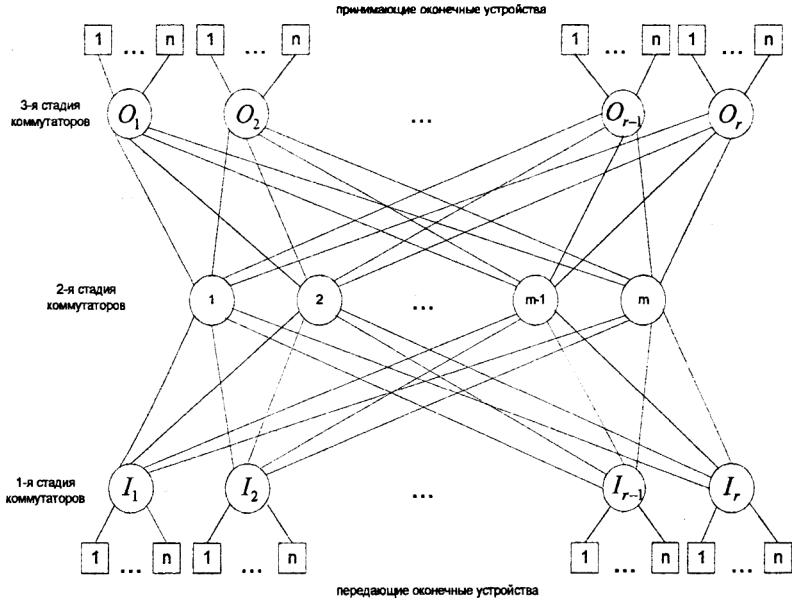


Рис 3. Схематическое изображение сети связи, построенной путём соединения коммутаторов по трёхкаскадной схеме Клоза.

Сеть состоит из трёх стадий простых полносвязных коммутаторов. На первой стадии располагается r коммутаторов, обозначим их $I_1, I_2, ..., I_r$ и будем называть входными коммутаторами. Каждый из них имеет по n входных и по m выходных портов. Первые используется для соединения с оконечными устройствами, а вторые с коммутаторами второй стадии. К каждому коммутатору подключается по n, а всего в сети имеется $n * r$ передающих оконечных устройств и столько же принимающих. Вторая стадия строится из m коммутаторов с r входными портами для подключения первой стадии и с r выходными портами для подключения третьей стадии. Таким образом, каждый входной коммутатор связан с каждым коммутатором второй стадии и наоборот, каждый коммутатор второй стадии связан с каждым

входным коммутатором. Третья стадия состоит из r коммутаторов, обозначим их $O_1, O_2, ..., O_r$ и будем называть выходными коммутаторами. Аналогично, каждый выходной коммутатор соединён с каждым коммутатором средней стадии. Так как трёхкаскадная схема Клоза полностью определяется тремя параметрами, то для рассматриваемой системы введём обозначение $\nu(m, n, r)$.

Тот факт, что некоторое передающее оконечное устройство $j \in U$ подключено ко входному коммутатору с номером k ($1 \le k \le r$) будем обозначать как $j \in I_k$, аналогичное обозначение будет использоваться и для принимающих оконечных устройств. Передающие хосты, соединённые с некоторым входным коммутатором I_k, будем обозначать как $i_{k,p}, p = 1,2,...n$, принимающие хосты, соединённые с некоторым выходным коммутатором O_l, будем обозначать как $o_{l,p}, p = 1,2,...n$.

Множество абонентов сети (обозначим его W) представляет из себя набор пар $(j,k), j \in U, k \in U, j \ne k$ отправитель - адресат, где отправитель является передающим, а адресат - принимающим оконечным устройством. Таким образом, узел $j \in U$ соединяется с первой стадией коммутационной схемы, а узел $k \in U$ с третьей. Пакеты, адресованные от узла - отправителя $j \in U$ к узлу - адресату $k \in U$, назовём пакетами пары (j,k).

На работу сети связи наложим некоторые достаточно общие ограничения, которые состоят в следующем. Множество пар абонентов отправитель - адресат не меняется в процессе функционирования системы или меняется очень редко. Появление новых пар и удаление старых происходит в некоторые фиксированные моменты времени, расположенные достаточно далеко друг от друга. Таким образом, мы будем предполагать, что имеет место квазистатическое расположение абонентов сети. Эти условия часто могут возникать при функционировании коммуникационной среды высокопроизводительных вычислительных комплексов и многих других сетей связи.

Предположим, что каждая пара использует для взаимодействия единственные путь, проходящий через некоторый коммутатор среднего уровня, поэтому существует ровно m вариантов (по количеству таких коммутаторов) для выбора направления движения пакета по сети связи. Назовём этот путь *виртуальным соединением* данных абонентов. Будем также полагать, что используется метод маршрутизации от источника,

при котором выбор маршрута установления соединения происходит в узле - отправителе и задаётся для каждого сообщения единственный раз, непосредственно перед его отправкой в сеть, что делает невозможным применение методов поиска свободного пути.

Относительно технологии передачи будем полагать следующее, если некоторая линия, которая выбрана для передачи сообщения, уже занята, то возникает блокировка соединения. Обработка такой ситуации осуществляется аппаратно и для протокола передачи соответствующая информация недоступна. Как станет ясно из постановки задачи, дальнейшие детали взаимодействия абонентов, такие как поведение пакета и абонентов при возникновении блокировки или характеристики потока входных вызовов, являются несущественными.

Несмотря на то, что для данной модели построения сети сложно найти примеры практической реализации, она выбрана нами для большей наглядности всех последующих построений. Кроме того, к ней с незначительными ограничениями, не влияющими на рассматриваемые в настоящей статье положения, может быть сведён ряд реально существующих систем (см. например [1]).

Определение критерия оптимальности, постановка задачи

Введём понятие оптимального функционирования сети Клоза с $n*r$ передающими хостами и $n*r$ принимающими хостами. Для этого рассмотрим сеть, построенную с использованием такой схемы соединения, при которой обеспечивается максимальная пропускная способность среди всех сетей с одинаковым количеством входов и выходов. Легко видеть, что полносвязная сеть удовлетворяет этому условию, так как в ней каждый вход непосредственно связан с каждым выходом, что обеспечивает существование отдельного пути для любой возможной пары абонентов.

Рассмотрим полносвязную сеть размерности $(n*r)\times(n*r)$ ($n*r$ входов и $n*r$ выходов). Пусть для данной сети и сети Клоза полностью совпадают все параметры, такие как количество абонентов, их расположение, характеристики входных потоков сообщений и технология взаимодействия оконечных устройств. Будем говорить, что сеть Клоза *функционирует оптимально*, если её пропускная способность совпадает с пропускной способностью соответствующей полносвязной сети.

Таким образом, задачей является определение условий, накладываемых на множество абонентов, при которых сеть Клоза способна функционировать оптимально в рассмотренном выше смысле.

Кроме того, если эти условия выполняются, необходимо уметь находить размещение виртуальных соединений внутри схемы, при котором соответствующий оптимум достигается. Такое размещение будем называть *оптимальным*.

Достаточное условие существования оптимального размещения виртуальных соединений

Обратимся сначала к модели ординарной коммутации каналов, при которой соединение может потребоваться только между свободной парой вход и выход. Для этой модели и трёхкаскадной схемы Клоза существует ряд результатов, которые можно обобщить и использовать для решения рассматриваемой в настоящей работе проблемы. В этом контексте интерес представляют работы Д. Слепяна, А. М. Дьюгида и В. Э. Бенеша [4]. В них исследуется свойство *перестраиваемости* трёхкаскадной схемы Клоза. Суть этого свойства для коммутационных схем состоит в том, что всегда можно перестроить существующие соединения таким образом, чтобы оказалось возможным установить новое соединение. Результаты данных исследований показали, что трёхкаскадная схема Клоза $\nu(m,n,r)$ является перестраиваемой, если $m \geq n$. Для случая виртуальных соединений это означает то, что если для каждого отправителя $j \in U$ существует единственный адресат $k \in U$ с которым тот взаимодействует и наоборот, то в случае $m \geq n$ существует оптимальное размещение виртуальных соединений внутри сети.

Для того, чтобы обобщить данный результат, фиксируем некоторого отправителя $i \in I_k, 1 \leq k \leq r$ и некоторый коммутатор третьей стадии $O_l, 1 \leq l \leq r$. *Пучком соединений* данного отправителя $i \in I_k$ с коммутатором O_l будем называть множество пар вида $\{(i,j) \in W : j \in O_l\}$. Множеством *входящих пучков* для некоторого коммутатора O_l назовём множество пучков соединений с данным коммутатором для всех отправителей W. Множеством *выходящих пучков* некоторого коммутатора I_k назовём множество пучков соединений, для всех отправителей $i \in I_k$ ($i \in W$). Верна следующая теорема.

Теорема 1. Достаточное условие существования оптимального размещения виртуальных соединения в трёхкаскадной схеме Клоза

$\nu(m,n,r)$ состоит в том, что для любого входного коммутатора имеется не более m выходящих пучков, а для любого выходного коммутатора не более m входящих пучков.

Доказательство практически без изменений повторяет доказательство теоремы Слепяна - Дьюгида и здесь не приводится.

Проанализируем условия теоремы 1. Во первых они не является необходимыми. Рассмотрим пример для схемы $\nu(2,2,3)$, изображённой на рисунке 4. В данном случае каждое входное оконечное устройство, подключённое к коммутаторам I_1 и I_2, взаимодействует с каждым оконечным устройством выходного коммутатора O_1. При этом на O_1 приходит 4 пучка соединений, однако оптимальное размещение виртуальных соединений существует и показано на рисунке.

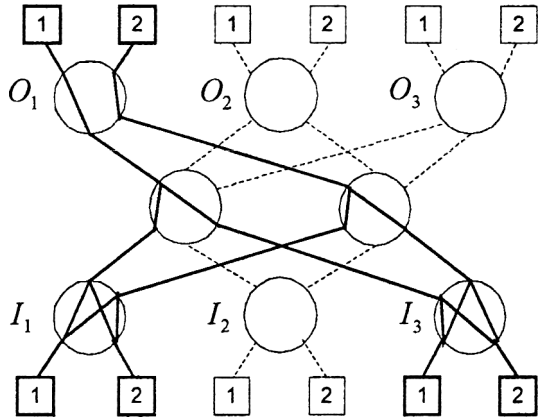


Рис 4. Пример размещения виртуальных соединений в схеме $\nu(2,2,3)$.

С другой стороны, ни условие на выходящие пучки коммутаторов $I_1, I_2, \ldots I_r$, ни условие на входящие пучки для коммутаторов $O_1, O_2, \ldots O_r$ в теореме 1 отбросить нельзя. На рисунке 5 изображён пример множества пар отправитель - адресат для той же схемы $\nu(2,2,3)$. Здесь условия теоремы не выполняются только для входного коммутатора I_1. Далее в статье мы покажем, почему оптимальное размещение виртуальных соединений в этой ситуации невозможно.

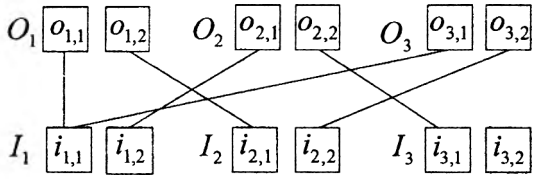


Рис 5. Пример множества W, для которого не существует оптимального размещения виртуальных соединений в схеме $\nu(2,2,3)$.

Представление множества конфликтующих виртуальных соединений

Основное отличие схемы Клоза от полносвязной сети состоит в том, что в первой могут возникать дополнительные блокировки соединений на второй стадии коммутаторов. Определим в каких случаях это возможно.

Пусть среди абонентов сети имеется 2 пары отправитель - адресат вида $(i_{k1,p1}, j_{l1,q1})$ и $(i_{k2,p2}, j_{l2,q2})$, где $1 \le k_1, k_2 \le r$ - номера входных коммутаторов, $1 \le l_1, l_2 \le r$ - номера выходных коммутаторов, $1 \le p_1, q_1, p_2, q_2 \le n$ - номера соответствующих входных и выходных хостов. Будем говорить, что виртуальные соединения данных пар *конфликтуют* между собой, если внутри сети Клоза возможно такое их расположение, при котором и то и другое проходят через один выходной порт коммутатора средней стадии, причём, для аналогичной ситуации в полносвязной схеме, данные пары не могут заблокировать друг друга.

Опишем все случаи, при которых возникают конфликтующие виртуальные соединения. Очевидно, что если $k_1 \ne k_2$ и $l_1 \ne l_2$, то виртуальные соединения данных двух пар никогда не пройдут через один и тот порт некоторого коммутатора среднего уровня. Если $k_1 = k_2, p_1 = p_2$, то при любом расположении виртуальных соединений внутри схемы блокировки между ними невозможны, так как у хоста $p_1(p_2)$ имеется единственный порт взаимодействия с сетью.

Аналогично, если $l_1 = l_2, q_1 = q_2$, то при любом расположении виртуальных соединений внутри схемы, блокировок избежать не удастся, так как соединения, адресованные на хост $q_1(q_2)$, должны будут проследовать через его единственный порт. Следовательно, те же самые блокировки будут возникать и в случае полносвязной сети. Таким образом, виртуальные соединения пар $(i_{k1,p1}, j_{l1,q1})$ и $(i_{k2,p2}, j_{l2,q2})$ являются *конфликтующими* тогда и только тогда, когда либо $k_1 = k_2 \wedge p_1 \neq p_2 \wedge (l_1 \neq l_2 \vee q_1 \neq q_2)$ либо $(k_1 \neq k_2 \vee p_1 \neq p_2) \wedge l_1 = l_2 \wedge q_1 \neq q_2$. То есть, либо отправители располагаются на различных портах одного и того же входного коммутатора и адресаты различны, либо отправители различны и адресаты располагаются на различных портах одного и того же выходного коммутатора.

Построим *граф состояния сети* следующим образом, вершинами графа служат пары абонентов (или виртуальные соединения), которые соединяются дугой тогда и только тогда, когда они конфликтуют. Тогда задача оптимального расположения абонентов сети Клоза $v(m,n,r)$ сводится к задаче правильной раскраски графа в m цветов. Все вершины (виртуальные соединения), окрашенные в один и тот же цвет, помещаются на один и тот же коммутатор средней стадии. Причём, для правильно раскрашенного графа будет автоматически получено оптимальное размещение виртуальных соединений сети.

В качестве примера рассмотрим конфигурацию абонентов, изображённую на рис. 5. На рис. 6 показан соответствующий этому множеству абонентов граф состояния сети.

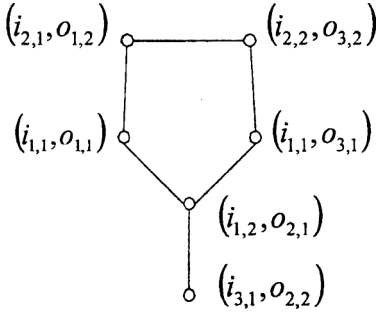


Рис 6. Пример графа состояния сети для множества W рисунка 5.

Проблема раскраски графов является хорошо изученной (см. например [5]), что даёт возможность применить к нашей задаче многие известные результаты и методы из теории хроматических графов.

Рассмотрим например граф, изображённый на рис. 6. Так как любой граф бихроматичен (2-хроматичен) тогда и только тогда, когда он не содержит нечётных простых циклов, то граф, изображённый на рис. 6, не может быть правильно окрашен в 2 цвета. Действительно, в нём содержится простой цикл $\{(i_{2,1}, o_{1,2}), (i_{2,2}, o_{3,2}), (i_{1,1}, o_{3,1}), (i_{1,2}, o_{2,1}), (i_{1,1}, o_{1,1})\}$, состоящий из пяти вершин. Следовательно, в примере рис. 5 также невозможно оптимальное размещение виртуальных соединений в схеме.

Эвристика для построения оптимального размещения соединений

Как известно, задача нахождения хроматического числа графа является NP- трудной, следовательно, весьма вероятно, что полиномиальный алгоритм раскраски отсутствует. Это порождает интерес к поиску методов приближённой раскраски, количество которых на данный момент очень велико. Остановимся на одном из них и на основании данного алгоритма построим эвристику, позволяющую получать оптимальное или близкое к оптимальному размещение виртуальных соединений в коммутационной схеме.

Рассмотрим алгоритм приближённой раскраски графов Ершова - Кожухина по книге [5]. В основе этого алгоритма лежит принцип склеивания вершин и сведение исходного графа с помощью такого склеивания к графу с меньшим числом вершин.

Назовём *окрестностью 1-го порядка* $R_1(v)$ вершины v множество вершин, смежных с вершиной v. *Склеивание* вершин v_1 и v_2 с окрестностями $R_1(v_1)$ и $R_1(v_2)$ соответственно есть преобразование, состоящее в удалении вершин v_1 и v_2 и добавлении новой вершины v, смежной со всеми теми вершинами, с которыми были смежны v_1 и v_2, так что $R_1(v) = R_1(v_1) \cup R_1(v_2)$. Это не приводит к появлению параллельных рёбер, так как вершины удаляются вместе с инцидентными рёбрами. В результате склеивания пары вершин получается граф с числом вершин на единицу меньшим, а также, возможно, с меньшим числом рёбер. Две вершины графа называются *соцветными*, если существует минимальная раскраска вершин графа, в которой они раскрашены в один цвет. Верны следующие утверждения.

Утверждение 1. В любом графе, отличном от полного, существует по крайней мере одна пара соцветных вершин.

Утверждение 2. Если граф G' получается из графа G склеиванием пары соцветных вершин, то $\chi(G') = \chi(G)$, где $\chi(G)$ ($\chi(G')$) - хроматическое число графа G (G').

Утверждение 3. Для всякого графа с хроматическим числом χ существует последовательность попарных склеиваний вершин, приводящая к полному χ-вершиннику.

Утверждение 4. Для любой вершины v, у которой окрестность 1-го порядка $R_1(v)$ не совпадает со всем множеством $X \setminus \{v\}$, в окрестности 2-го порядка имеется по крайней мере одна соцветная вершина. Здесь X - множество вершин графа G.

В утверждениях 1 - 4 по сути сформулирован алгоритм приближённой раскраски графа, который состоит в последовательном выборе и склеивании вершин до тех пор, пока не образуется полный граф. В утверждении 4 говорится о том, что вершину -претендента для склеивания можно искать во множестве вершин второго порядка. Нерешённым, однако, остаётся вопрос, по какому принципу выбирать данную вершину, если имеется несколько вершин второго порядка.

Интуитивно ясно, что хороший алгоритм должен выбирать пару вершин графа для склеивания таким образом, чтобы максимально предотвращать формирование полного графа. Для этого, например, в работе [6] предлагается вычислять для всех пар несмежных вершин число общих смежных вершин и выбирать для склейки пару с наибольшим таким числом.

Будем за один раз склеивать не одну, а несколько вершин графа состояния сети, то есть распределять сразу несколько виртуальных соединений на некоторый промежуточный коммутатор. Рассмотрим пучок соединений некоторого хоста источника $i \in I_k$ с коммутатором O_l. Все соединения этого пучка могут быть окрашены в одинаковый цвет, так как в графе состояния их, очевидно, не соединяет ни одно ребро. Кроме того, если для двух вершин пучка v_1 и v_2 в графе состояния существует путь, который их соединяет, то $v_1 \in R_2(v_2)$ и $v_2 \in R_2(v_1)$, а значит, эти вершины являются пригодными кандидатами для склеивания. За один шаг предлагается выбирать по одному пучку виртуальных соединений с каждого входного коммутатора I_k, при этом на каждый раз нужно стараться сделать так,

чтобы суммарное количество рёбер, которое исчезнет из графа при склеивании было максимально среди всех таких наборов.

В силу симметричности схемы и определения конфликтующих соединений, можно дополнительно на каждом шаге менять местами входы и выходы сети и отправителей с адресатами в каждой паре множества W. Получившуюся при этом сеть будем называть *обратной*. Граф состояния в обратной сети, как следует из определения, остаётся прежним. Далее необходимо проделывать те же действия со вновь получившимися пучками и находить максимум суммарного количества исчезающих при склеивании рёбер уже из двух вариантов: исходной сети и обратной сети.

Сформулированная эвристика ни в коем случае не претендует на завершённость и требует проведения дальнейших исследований и экспериментов как с реально существующими системами, для обоснования применимости подхода вообще, так и с моделями, для получения более качественных оценок и формирования строгого алгоритма.

Сценарий возможного применения обсуждаемого подхода

В качестве примера возможного применения предложенного метода управления соединениями в реальных сетях передачи данных рассмотрим высокопроизводительный вычислительный комплекс, построенный по кластерной архитектуре. Суть данного метода построения системы заключается в том, что вычислительное поле строится из множества вычислительных модулей, соединённых высокоскоростной сетью связи. Каждый модуль представляет собой отдельную ЭВМ, работающую под управлением собственной операционной системы. Сеть связи обеспечивает возможность обмена данными между вычислительными модулями в процессе выполнения задачи, при этом в качестве способа взаимодействия может использоваться, например, интерфейс передачи сообщений (MPI). Так как топология Ч. Клоза широко применяется при создании подобных систем, то будем полагать что сеть передачи данных построена с её использованием.

В процессе работы в систему поступают задачи, каждая из них представляет собой набор параллельно работающих ветвей. Параллельная ветвь может занимать один вычислительный модуль (или один процессор, если вычислительный модуль представляет из себя многопроцессорную систему) и обмениваться информацией с другими

ветвями в процессе счёта. Таким образом, для каждой задачи существует некоторая топология взаимодействия.

Предположим, что информация о топологии взаимодействия каждой параллельной задачи известна непосредственно перед запуском задачи. Данную информацию может определить пользователь в некотором паспорте, передаваемой системе управления заданиями, или она может быть получена каким либо другим способом. Тогда выполнение условий теоремы 1 гарантируют существование, а предложенный подход позволяет построить такое размещение виртуальных соединений, при котором сеть передачи данных будет функционировать оптимально, то есть обеспечивать такие же характеристики передачи (задержку сообщения, вероятность блокировки соединения и т. д.), что и полносвязная сеть.

Таким образом, в системе управления заданиями необходимо реализовать модуль, который управляет виртуальными соединениями в системе. Основной его функцией является размещение и переразмещение соединений при запуске одних задач и окончании счёта других, а в основе алгоритма работы лежит рассматриваемый в данной работе подход.

Заключение

Обозначим основные положения, рассмотренные в настоящей статье. Нами было получено достаточное условие существования оптимального (относительно полносвязной сети) размещения виртуальных соединений пар абонентов. Для построения данного размещения предложен эвристический подход с привлечением теории и методов хроматических графов. Настоящий подход может использоваться при решении задачи управления маршрутами в сетях передачи данных, построенных с использованием топологии Ч. Клоза. Его преимущество состоит в относительной простоте реализации, так как при выполнении условий теоремы 1, не требуется производить никаких дополнительных измерений и вычислений параметров сети, определяемых её текущим состоянием и входной нагрузкой. Кроме того он не зависит от особенностей используемой технологии передачи, так как опирается только на топологические свойства схемы соединения.

Литература

1. Guide to Myrinet 2000 Switches and Switch Networks, 2001. <http://www.myri.com/myrinet/m3switch/guide>

2. В. В. Корнеев, Параллельные вычислительные системы, М.: Нолидж, 1999

3. C. Clos. A Study of Non-Blocking Switching Networks. "Bell System Tech, J", 32, 406-424 (1953)

4. В. Э. Бенеш, Математические основы теории телефонных сообщений: Пер. с англ./ Под ред. И. Н. Коваленко. - М.: Связь, 1968. - 291 с.

5. В. А. Евстигнеев, применение теории графов в программировании: Под ред. А. П. Ершова. - М. «Наука», 1985.

6. Dutton R. D., Bingham R. C. A new graph coloring algorithm. - The Computer J., 1981, 24, № 1, p. 87 - 91.

Павлов Б.М., **Новиков М.Д.**

Пакет программ для компьютерного моделирования пространственно-временных нелинейных процессов

Введение

Реальные нестационарные процессы в системах живой и неживой природы вдали от равновесия могут приводить к *самоорганизации* – спонтанному образованию сложных структур, упорядоченных в пространстве–времени. Наблюдения и теоретические исследования показали, что самоорганизация возможна лишь в открытых диссипативных системах, обменивающихся энергией или веществом с внешней средой. Такие системы являются активными, т.е. способными в определённых условиях к генерации автоколебаний, образованию стационарных или пульсирующих пространственных конфигураций и распространению различных волн.

Несмотря на большое разнообразие биологических, экологических, химических и физических активных систем (сред), изучение их особенностей может быть сведено к исследованию решений небольшого набора простейших нелинейных математических моделей, называемых базовыми. Ценность базовой модели заключается в том, что с её помощью можно качественно предсказать пространственно-временное поведение исследуемого процесса: развитие различных неустойчивостей, возникновение бифуркаций; можно также определить зависимость процесса от параметров и выбора начальных и граничных условий.

Конструирование и подробное исследование базовых математических моделей различных классов, описывающих нелинейные колебания и волны, являются задачами нелинейной динамики и её молодой ветви – синергетики, ныне рассматриваемой в качестве общей теории самоорганизации в активных средах различной природы [1-3]. Из-за неизбежного присутствия нелинейностей в базовых моделях их подробное качественное и количественное исследование невозможно (за крайне редкими исключениями) без применения компьютера, численных методов и программных средств.

Изучение свойств различных базовых моделей в синергетике (детерминированного хаоса, странных аттракторов, бегущих фронтов и импульсов, диссипативных структур и т.п.) очень привлекательно не

только с точки зрения получения нового знания в этой области, но и для вычислительных и лабораторных практикумов, курсовых и дипломных работ при обучении прикладной математике и математическому моделированию с применением численных методов, компьютерной графики и анимации математических образов. Самоорганизация во времени (выход на аттракторы эволюции) изучается на основе точечных моделей в форме систем разностных уравнений (РУ) и систем обыкновенных дифференциальных уравнений (ОДУ) невысокой размерности ($n \leq 4$).

Для численного исследования процессов самоорганизации во времени на факультете ВМК МГУ в середине 90-х годов прошлого века была разработана и реализована на персональном компьютере концепция обучения «по образцу-эталону», основанная на технологии вычислительного эксперимента. Для проведения вычислительных экспериментов была создана учебно-исследовательская среда (УИС) с набором тщательно протестированных нелинейных моделей РУ и ОДУ (включая уравнения с так называемыми «запаздываниями»), а также с необходимым интеллектуальным инструментарием для идентификации различных динамических режимов. УИС была реализована в виде интерактивного пакета прикладных программ «Нелинейные динамические системы» (НДС) на языке Паскаль для ПК типа IBM-PC в операционной системе DOS. Подробное описание пакета содержится в учебном пособии [4], где приведены также формулировки индивидуальных заданий для студентов, необходимые теоретические сведения из нелинейной динамики и словарь терминов.

В данной статье дана постановка задачи численного исследования процессов самоорганизации во времени и пространстве для математических моделей в форме систем уравнений в частных производных типа «реакция-диффузия». Описана новая комфортная УИС для эффективного проведения вычислительных экспериментов (ВЭ) с такими моделями; основу УИС составляет пакет прикладных программ «Диссипативные структуры» (кратко – пакет «DISS»). Приведены примеры протестированных базовых моделей и некоторые результаты численного исследования, выполненного с помощью пакета «DISS».

Работа была доложена на Ломоносовских чтениях МГУ в апреле 2002 г.

Постановка начально-краевой задачи для систем типа «реакция-диффузия». Численная методика исследования

Наиболее популярными для объяснения многих нелинейных феноменов и пространственно-временной самоорганизации активных сред являются одномерные по пространству базовые модели Тьюринга, Лотки и Вольтерры, Жаботинского, Мейнхарда и Гирера, Фитц Хью и Нагумо, «брюсселятор» Пригожина и Лефевра. Каждая из этих моделей описывается системой двух или трёх параболических уравнений типа «реакция-диффузия». Поэтому из множества распределённых моделей, более сложных, чем сосредоточенные (точечные) модели, мы на первом этапе выбрали для включения в пакет программ DISS именно модели класса «реакция-диффузия», причём в настоящее время возможно исследование одномерных по пространству уравнений с не более, чем тремя компонентами. Даже такие простые модели позволяют изучать разнообразные автоволновые процессы и диссипативные структуры, возникающие при тех или иных начальных и краевых (граничных) условиях и значениях параметров.

Достаточно общую типовую математическую модель из класса «реакция-диффузия» составляет следующая начально-краевая задача для нестационарной системы квазилинейных уравнений в частных производных параболического типа с параметрами $\{\bar{A}, \bar{D}, \bar{\varepsilon}\}$:

$$\begin{aligned} \varepsilon_1 U_t &= D_1 U_{xx} + Q_1(U,V,W,\bar{A}), \\ \varepsilon_2 V_t &= D_2 V_{xx} + Q_2(U,V,W,\bar{A}), \\ \varepsilon_3 W_t &= D_3 W_{xx} + Q_3(U,V,W,\bar{A}). \end{aligned} \qquad (1)$$

Здесь: $U(x,t)$, $V(x,t)$ и $W(x,t)$ – компоненты (искомые решения), $U_t \equiv \partial U/\partial t$, $U_{xx} \equiv \partial^2 U/\partial x^2$ и аналогично для производных V и W. Приведена трёхкомпонентная модель; в двухкомпонентной (однокомпонентной) модели отсутствует третье (второе и третье) уравнение и переменная W (V и W). Величины Q_i задают «реакции» и являются нелинейными функциями (источниками или стоками) своих аргументов, $\bar{A}$ – вектор числовых параметров, $\bar{D}=(D_1,D_2,D_3)$ – постоянные неотрицательные коэффициенты одномерной диффузии по x-координате, $\bar{\varepsilon}=(\varepsilon_1,\varepsilon_2,\varepsilon_3)$ – специально выделенные «малые параметры», $0<\varepsilon_i\le 1$ ($\varepsilon_i=1$ означает отсутствие малого параметра); $t\ge 0$, x изменяется на некотором отрезке $[L_1,L_2]$, причём обычно $L_1=-L_2$ или $L_1=0$. В дальнейшем для определённости будем считать, что x изменяется на симметричном отрезке $[-L,L]$.

Начальные условия: при t=0 задаются функции $U_0(x)$, $V_0(x)$ и $W_0(x)$ такие, что

$$U(x,0)=U_0(x),V(x,0)=V_0(x) \text{ и } W(x,0)=W_0(x),-L\le x\le L,L<\infty. \quad (2)$$

Граничные условия: при t>0 на концах отрезка [–L,L] задаются линейные комбинации

$$\alpha_1 U+\beta_1 U_x=\gamma_1,\ \alpha_2 V+\beta_2 V_x=\gamma_2,\ \alpha_3 W+\beta_3 W_x=\gamma_3, \quad (3)$$

где α_i, β_i и γ_i (i=1,2,3) – заданные числа, а значения U и U_x берутся при $x=\pm L$. Частными случаями такой общей записи являются граничные условия Дирихле (ГУ-1) при $\alpha_i=1,\beta_i=0$ и граничные условия Неймана (ГУ-2) при $\alpha_i=0$, $\beta_i=1$. Если $\alpha_i=0$, $\beta_i=1$ и $\gamma_i=0$, то это так называемые нейтральные условия или условия непроницаемости границ конечного отрезка.

Заметим, что всех при $D_i\equiv 0$ имеем в качестве частных случаев одно-, двух- и трёхкомпонентные точечные системы ОДУ, для исследования которых эффективен упоминавшийся выше пакет «НДС». В частности, с его помощью можно численно находить для распределённой системы (1) однородные по x решения и изучать их устойчивость: это стационары по t (равновесия), периодически колеблющиеся по t решения, квазипериодические и хаотические решения.

Как известно из теории автоволновых процессов (см., например, монографии [5-12]), уравнения типа (1) для одной компоненты позволяют исследовать лишь бегущие по x устойчивые фронты (БФ) в виде волны переключения (переброса, перестройки). Двухкомпонентные системы (1) позволяют, кроме БФ, наблюдать и изучать деление фронтов, бегущие импульсы (БИ), автоколебания однородных и неоднородных по x распределений, стоячие волны, локальные пульсирующие и самодостраивающиеся контрастные структуры. Трёхкомпонентные системы (1), кроме перечисленных возможностей, пригодны для изучения сложных «дышащих» по t и x диссипативных структур, а также ведущих центров (ВЦ) – источников расходящихся волн. Все эти типовые автоволновые процессы неоднократно наблюдались и воспроизводились в лабораторных экспериментах - биологических, физических и химических и изучались в численных экспериментах (см, например, [5,8,11]) .

Большое разнообразие автоволновых процессов проистекает из-за всевозможного сочетания значений параметров модели $\{\overline{A},\overline{D},\overline{\varepsilon}\}$, характера нелинейной функциональной зависимости $Q_i(U,V,W)$, значения L («размер» системы), а также из-за вида используемых начальных и граничных условий. Если коэффициенты ε_i и D_i различаются между собой по величине на несколько порядков, то существенно различаются и скорости изменения соответствующих

компонент U,V и W во времени и пространстве. Например, в случае двухкомпонентной модели, когда параметр $\sigma=D_1/D_2$ является малым, возникают контрастные диссипативные структуры (КДС), которые содержат по x-координате чередующиеся участки резкого («пички») и плавного изменения по крайней мере одной из компонент. В практическом отношении случай контрастных структур является важным и имеет широкую область применимости (например, КДС в плазме и полупроводниках [11]).

Для численного решения начально-краевой задачи (1), (2), (3) будем использовать метод конечных разностей. Для этой цели в области изменения независимых переменных $\{t>0,\ -L\le x\le L\}$ вводится регулярная сетка равноотстоящих узлов (t_n,x_k), где $t_{n+1}=t_n+\Delta t$, $x_{k+1}=x_k+\Delta x$, n=0,1,2,..., k=0,1,2,...,M, $M=2L/\Delta x$. Далее выполняется аппроксимация производных искомых функций U,V и W в узлах этой сетки с той или иной степенью точности относительно шагов сетки Δt и Δx. Нами была выбрана и прошла испытания на ряде хорошо известных базовых моделей типа «реакция-диффузия» явная разностная симметричная схема  , в шаблоне которой используются три узла на предыдущем n-ом временном слое и один узел на следующем (n+1)-м временном слое. Производные по t аппроксимировались с первым порядком точности, вторые пространственные производные – со вторым порядком точности. Первые производные U_x, V_x и W_x в граничных точках $x=\pm L$ аппроксимировались со вторым порядком точности по трём соседним точкам на (n+1)-м слое: $U_x=1/(2*\Delta x)*(-3*U_0+4*U_1-U_2)$ на левой границе и $U_x=1/(2*\Delta x)*(U_{M-2}-4*U_{M-1}+3*U_M)$ на правой границе (для V и W аналогично). Эта явная схема является условно устойчивой: для рассматриваемой системы (1) критерием устойчивости служит неравенство:

$$\Delta t\le C*(\Delta x)^2/\max\{D_i\},\ i=1,2,3, \qquad (4)$$

где $C\le 1$ – некоторая константа. Невыполнение этого условия приводит к возникновению вычислительной неустойчивости, не имеющей никакого отношения к возможным физическим неустойчивостям при определённых сочетаниях параметров $\{\bar{A}, \bar{D}, \bar{\varepsilon}\}$ в изучаемой модели нелинейного процесса. Наши ВЭ показали, что приемлемая точность численных результатов достигается с не очень большим (50-400) количеством узлов по x-координате (L~2–20) и при соблюдении критерия (4) с C=0.5. В таких условиях можно обнаружить практически все известные особенности нелинейного поведения моделей при наличии реальных малых параметров ε_i и σ. Хорошо воспроизводятся сложные ситуации динамического характера (резкие релаксационные колебания по t и узкие области по x с большими

градиентами решений – «пички»). Подчеркнём, что объёмы вычислений на шаге по времени в пакетах «DISS» и «НДС» соотносятся примерно как N:1, где N – число разбиений отрезка [–L,L].

Назначение и структура пакета DISS

Пакет предназначен для проведения вычислительных экспериментов с математическими моделями распределённых динамических систем. Первая очередь интерактивного пакета позволяет исследовать свойства нестационарных одномерных моделей типа «реакция-диффузия» с одной, двумя или тремя компонентами вектора решения (U,V,W). Пакет «DISS» организован по «модульному» принципу, как и пакет «НДС», используемый в вычислительном практикуме факультета ВМК более 5 лет. Вычислительные модули пакета включают в себя процедуры расчёта сеточных функций на (n+1)-м временном слое. Набор исследовательских инструментов анализа состоит из модулей: определения периода автоколебаний по t и периода структур по *x*; построения таблиц, а также статических и динамических графиков, обеспечивающих визуализацию и анимацию всевозможных диссипативных структур, бегущих фронтов и импульсов, ведущих центров, «дышащих» структур и проч. Сервисные модули пакета предназначены для создания, пополнения и чистки банка математических моделей (БММ), банка эталонных решений (БЭР), для автодемонстрации накопленных эталонных решений в форме слайдов и для печати на принтере картинок с дополнительной текстовой информацией.

Пакет «DISS» предоставляет пользователю *комфортную* среду для проведения ВЭ. Под комфортностью мы понимаем реализацию следующих естественных пожеланий пользователя:

1. Задание с отображением на экране ПК любого допустимого вида уравнений модели, начальных и граничных условий, значений параметров с помощью клавиатуры и (или) мыши.
2. Гибкий режим проведения текущего ВЭ с его прерыванием через определённое число шагов по времени для анализа промежуточных численных решений по *x* и t; предусмотрены возможности изменения в ходе ВЭ шага по t и значений некоторых параметров, а также возврата к более раннему t для нового интегрирования с изменённым шагом.
3. Интуитивно понятный интерфейс «пакет-пользователь», включающий в себя, в частности, сообщения о некорректных действиях пользователя и об ошибках в процессе счёта (переполнение и проч.)
4. Удобные средства работы с БММ и БЭР.

Интерфейс пользователя с пакетом DISS организован в форме трёхуровневого меню. Первые 2 уровня меню представлены на рис.1. При выборе какого-либо пункта из серии «Графики, таблицы» открывается меню третьего уровня: предлагается выбрать либо вывод значений U,V,W, либо вывод экстремумов U,V,W.

При выборе любого пункта меню открывается соответствующее диалоговое окно. Опишем некоторые диалоговые окна. Так, при выборе пункта «Выбор системы из списка» на экран выводится окно с перечнем всех имеющихся в БММ моделей; щелчком мыши можно выбрать любую из них для последующей работы (счёта, вывода графиков и проч.). При выборе пункта «Ввод новой системы» открывается диалоговое окно ввода вида системы и её параметров (Q_i, D_i , $\overline{A}$), начальных распределений (функций $U_0(x)$, $V_0(x)$ и $W_0(x)$), граничных условий (коэффициентов α_i, β_i и γ_i,), шага по t и количества разбиений отрезка [-L,L]. Для начала счёта требуется задать количество слоёв интегрирования по t. При выводе таблицы или графика предлагается ввести значения t и x и указать, какие из компонент U,V и W следует выводить.

В режиме показа графика нажатие клавиши F3 позволяет скопировать графический экран в файл на диске (пополнить БЭР); клавиша F5 вызывает печать графика на принтере. В режиме «Автодемонстрация» можно листать вперёд и назад файл с накопленными картинками из БЭР.

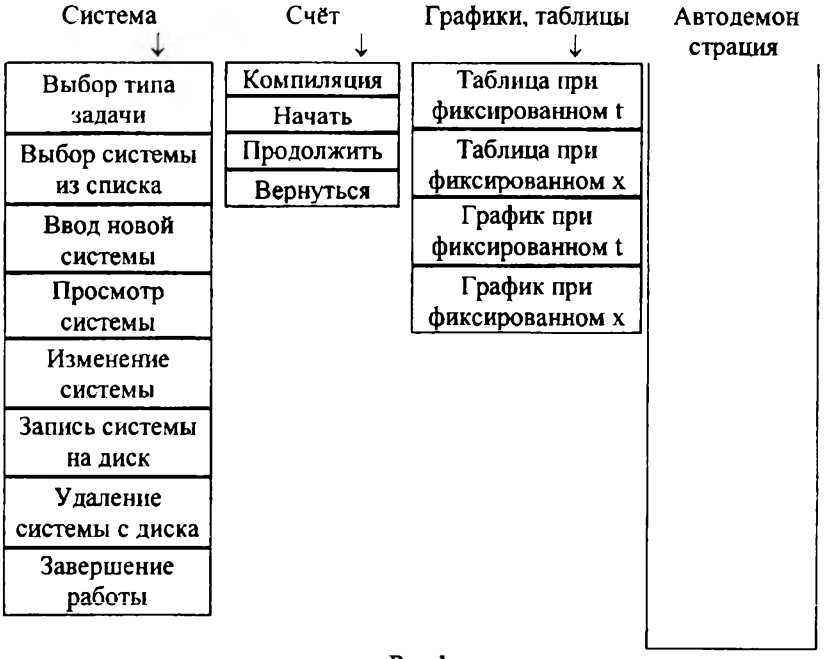


Рис.1.

Пакет «DISS», в отличие от пакета «НДС», реализован в инструментальной среде DELPHI (версия 5) и работает под управлением ОС Windows-95/98/2000/NT. Основной исполняемый файл – DISS.EXE; он занимает объём около 600Кб. Реализованы 2 способа вычисления функций Q_i, U_0, V_0 и W_0: в режиме интерпретации и после компиляции (второй способ в несколько раз быстрее). По умолчанию используется режим интерпретации. Для компиляции пользователь должен выбрать одноимённый пункт меню; при этом производится компиляция текущей системы. После выбора другой системы надо выполнить компиляцию вновь (иначе будет установлен режим интерпретации). Файл DISSPRED.EXE, исполняющий компиляцию, занимает объём около 300Кб. Если на ПК не установлена система DELPHI, то для работы файла DISSPRED.EXE требуется DELPHI-компилятор, занимающий объём около 15Мб.

Приведём в форме таблицы сведения о некоторых популярных моделях, прошедших апробацию в ВЭ с помощью пакета «DISS». Обозначения: АВ – автоволны, АС – автосолитон, ДС – диссипативная структура, БАС – бегущий автосолитон, ПАС – пульсирующий автосолитон, СДС – стационарная ДС, АОС – автоколебания однородных состояний (однородные колебания), АНС – автоколебания

неоднородных состояний (неоднородные колебания), ПМ – пейсмейкер, ВЦ – ведущий центр.

На рис. 2 - 4 представлены графики решений для некоторых моделей с комментариями.

Литература

1. Николис Г., Пригожин А. Самоорганизация в неравновесных системах. М.: Мир, 1979.
2. Эбелинг В. Образование структур при необратимых процессах. М.: Мир, 1979.
3. Хакен Г. Синергетика. М.: Мир, 1980.
4. Павлов Б.М., Новиков М.Д. Автоматизированный практикум по нелинейной динамике (синергетике). М.: Изд-во МГУ, 2000.
5. Жаботинский А.М. Концентрационные колебания. М.: Наука, 1974.
6. Ланда П.С. Нелинейные колебания и волны. М.: Наука, 1997.
7. Лоскутов А.Ю., Михайлов А.С. Введение в синергетику. М.: Наука, 1990.
8. Полак Л.С., Михайлов А.С. Процессы самоорганизации в физико-химических системах. М.: Наука, 1983.
9. Романовский Ю.М., Степанов Н.В., Чернавский Д.С. Математическая биофизика. М.: Наука, 1984.
10. Васильев В.А., Романовский Ю.М., Яхно В.Г. Автоволновые процессы. М.: Наука, 1987.
11. Кернер Б.С., Осипов В.В. Автосолитоны. М.: Наука, 1991.
12. Свирежев Ю.М. Нелинейные волны, диссипативные структуры и катастрофы в экологии. М.: Наука, 1987.

	Модель (ссылка, комментарий)	Вид «реакций» Q_i, параметры модели	АВ и АС, обнаруженные в ходе ВЭ
.1	***Брюсселятор*** Пригожина и Лефевра: [1,6,8, 11],в [11] опечатка в уравнении 1.13, с.24	$Q_1=A-(1+B)*U-V*U^2$ $Q_2=U*(B-U*V)$ Параметры:A,B, D_1,D_2.	СДС широкие и пичкового типа, самодостройка СДС с увеличением значения L
.2	***Жаботинский***: [5,11]	$Q_1=(B+U*(1-V*(2+(U-1)^2)))/\varepsilon$ $Q_2=-A*V+U*(1-V)$ Параметры: A,B,ε,D_1,D_2	Широкие и узкие АС, СДС.
.3	***Майнхардт-Гирер***: [9,11]	$Q_1=B+A*U^2/V-U$ $Q_2=C*U^2-V$ Параметры: A,B,C,D_1,D_2	СДС пичкового типа
. 4	***Вольтерра*** (одна из модификаций) [12]	$Q_1=U*(-V+(A+B)*U-C*U^2)$ $Q_2=V*(U-D-E*V)$ Параметры: A,B, C, E, D_1,D_2	СДС – периодические и хаотические
.5	***Фитц Хью-Нагумо*** (ФХН): [6,7,10,11]; в [11] имеются опечатки в формуле Q_2 с.159.	$Q_1=-U^3/3+U-V$ $Q_2=5/3*U+2*A-2*V$ Параметры: A,D_1,D_2	БИ, ВЦ при $D_2=0$. В режиме ВЦ должно быть A=0.1505 вместо A=1.505 в [11], с.161 при $D_2\neq0$.
.6	***Ринцель-Келлер*** – упрощённая модель ФХН, Q_1 – кусочно-линейная функция [7,8,11]	$Q_1=-U+H(U,A)-V$ $Q_2=B*U-V$, где $H(U,A)=1$, если $U\geq A$ и 0, если $U<A$. Параметры: A,B, D_1,D_2; $D_2\equiv0$.	БАС, ПМ, ПАС, ВЦ, АОС, АНС
.7	***Васильев – Заикин*** – расширенная модель брюсселятора [10].	$Q_1=W*U+U^2*V-(1+B)*U$ $Q_2=U*(B-U*V)$ $Q_3=A+C*U-W$ Параметры: A,B,C, D_1,D_2,D_3	СДС, ВЦ

Брюсселятор

Параметры - A=0.9;B=2;C=0.1;K=10;X_0=0

Q_1=A–(1+B)*U–V*U^2 ,Q_2=U*(B–U*V)

D_1=0.002, D_2=0.2

U(x,0)=A+C*exp(-K*(x-X_0) *(x-X_0)), V(x,0)=B/A

Границы изменения x:[-4,4]; t=30

На левой границе: U_x=0, V_x=0

На правой границе: U_x=0, V_x=0

Шаг по x – 0.1, шаг по t – 0.002.

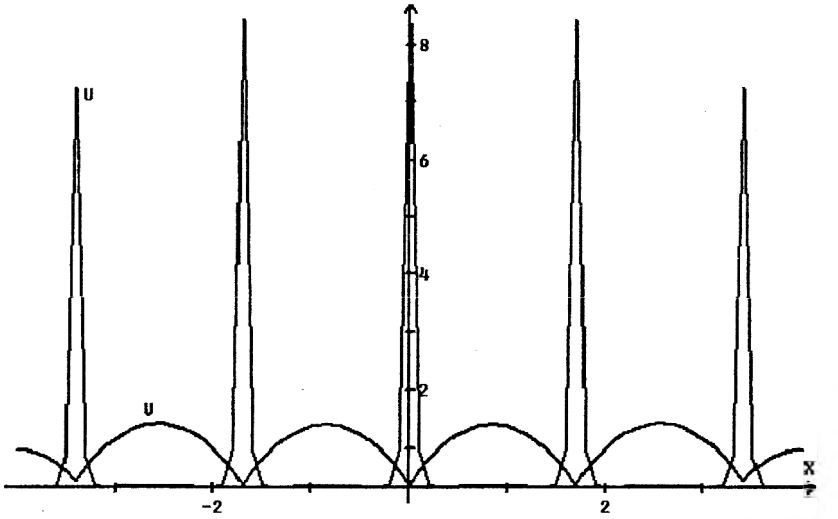


Рис.2

Распределённая модель «Брюсселятор» демонстрирует на отрезке [–4,4] контрастную ДС, симметричную относительно x=0. Эта СДС возникла в три этапа: сначала стабилизировался центральный пичок U при x=0, затем одновременно возникли пички U слева и справа от него примерно с теми же максимумами и, наконец, возникли пички U с меньшими максимумами у границ отрезка (режим самодостройки диссипативной структуры).

Свирежев

Параметры-A=35;B=16;C=1;K=0.1;G=0.5; V_0=0.1; M_0=1; M_1=0.4

Q_1=U*(-V+(A+B*U-C*U*U)/9), Q_2=V*(U-M_0-M_1*V)

D_1=0.0001, D_2=1

U(x,0)=A+A*G*cos(π*K*x), V(x,0)=B+V_0*cos(π*K*x)

Границы изменения x:[0,5], t=10

На левой границе: U_x=0, V_x=0

На правой границе: $U_x=0$, $V_x=0$
Шаг по x - 0.1, шаг по t - 0.0005

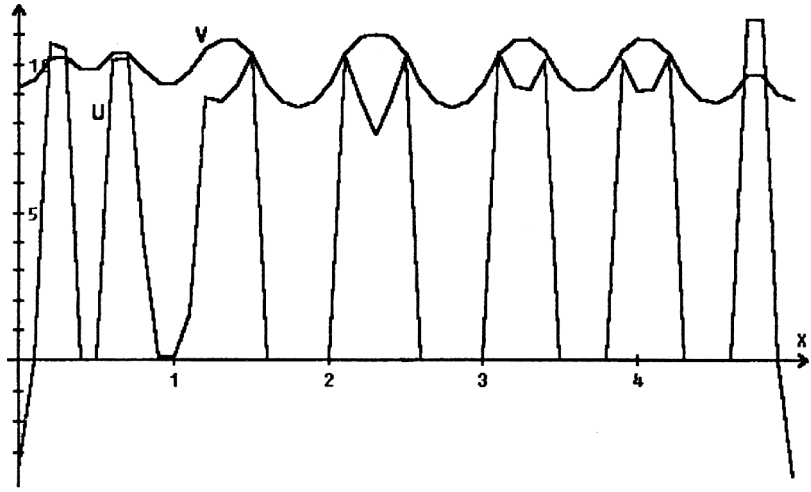


Рис.3

Распределённая модель «хищник-жертва», описанная в [12], демонстрирует при заданных выше условиях сложную стационарную ДС, которая не является ни периодической, ни симметричной; её можно отнести к хаотической. ВЭ показали, что в тех же условиях, но при увеличении коэффициента диффузии D_1, хаотичность, «изрезанность» распределений U (жертвы) и асимметрия графиков по x уменьшаются. Модель имитирует нередко наблюдаемую в природных ареалах «пятнистость» расселения (то много, то мало) хищников и их жертв.

Кернер-Осипов

Параметры - A=0.1505; B=0.02; E=–1; K=20; X_0=0; U_0=–0.985; V_0=–0.5;C=0.125

Q_1=(U-U*U*U/3-V)/B, Q_2= –E* (5*U/3-2*V+2*A)

D_1=0.032, D_2=0.034

$U(x,0)=U_0+C*\exp(-K*(x-X_0)*(x-X_0))$, $V(x,0)$= –0.670333333

Границы изменения x:[-8,8], t=15

На левой границе: $U_x=0$, $V_x=0$

На правой границе: $U_x=0$, $V_x=0$

Шаг по x – 0.08, шаг по t – 0.0025

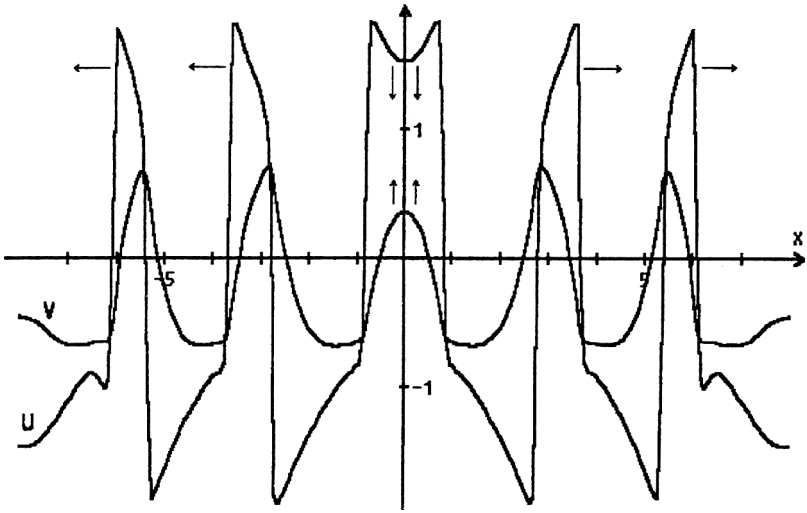


Рис.4

Эта модель демонстрирует при заданных условиях бегущие импульсы, распространяющиеся влево и вправо из центральной области [-1,1], в которой U и V совершают периодические вертикальные колебания; БИ «гибнут» у границ расчётной области [-8,8]. Это режим стабильного ведущего центра, впервые найденный авторами [11] для двухкомпонентных (а не трёхкомпонентных) систем при $D_2 \neq 0$. Однако эти авторы не смогли найти режим ВЦ для более интересного случая с $D_2=0$ (модель ФХН), что удалось сделать нам с помощью пакета DISS.

Гамаюнов Д. Ю.

Современные некоммерческие средства обнаружения атак

В данной статье приводится обзор и сравнительный анализ нескольких существующих некоммерческих систем обнаружения атак (СОА) на компьютерные системы и сети. Проведен сравнительный анализ 7 систем по определенному набору критериев, который разрабатывался специально для сравнения и оценки качественных характеристик СОА.

Введение

Последнее десятилетие ознаменовалось бурным развитием компьютерных сетей, что породило множество проблем, связанных с безопасностью информационных ресурсов. В конце 70-х – начале 80-х годов прошлого века стали появляться первые системы обнаружения атак на вычислительные системы. В 90-х годах происходил бурный рост количества исследований и действующих систем в данной области. Сегодня можно насчитать более сотни разных систем обнаружения атак, большинство из которых в настоящее время не развиваются и не поддерживаются.

Несмотря на высокую активность в исследовании компьютерных атак, на сегодня нет теории обнаружения атак. Нет даже общепринятого определения самого понятия атаки. Поэтому формальные методы обнаружения атак проработаны недостаточно для широкого использования в реальных системах. Методы обнаружения, используемые сегодня в коммерческих и некоммерческих СОА, можно назвать эвристическими. Они все используют некоторые априорные предположения о том, что есть атака, какое поведение объекта в сети можно считать нормальным, а какое подозрительным.

Коммерческие системы большей частью используют один и тот же эвристический метод обнаружения, основанный на экспертном подходе, когда система анализирует наблюдаемое поведение объектов в сети на основе существующей у нее базы описаний известных атак. Эти описания строятся на основе знаний экспертов в области сетевой безопасности. При этом для коммерческих систем характерны высокая начальная стоимость и высокая стоимость поддержки.

Для экспериментальных систем характерно использование в большей степени формализованных методов обнаружения атак, которые

используют формальную модель атаки и пытаются приблизить процесс ее обнаружения к полной автоматизации. Используются такие разделы математики как нейронные сети, сети Петри, конечные автоматы.

В последние годы активно начали разрабатываться и использоваться некоммерческие системы обнаружения атак. Часть из них разрабатываются в университетах и «выросли» из исследовательских систем. Прочие являются разработками компаний, которые работают по распространенному сегодня принципу открытого программного обеспечения. В данной работе приводится обзор современных некоммерческих СОА. Обзор не претендует на полноту, его основная цель – дать читателю общее представление о возможностях некоммерческих СОА, помочь лучше сориентироваться на рынке коммерческих систем.

Исследованные системы обнаружения атак

Всего рассмотрено 7 систем обнаружения атак. В табл. 1.1. приведена краткая информация по каждой из них.

Название системы	Производитель	Ссылки
AAFID	Purdue University, West Lafayette, IN, USA	http://www.cs.purdue.edu/coast/projects/autonomous-agents.html
ASAX	University of Namur, Belgium	http://www.ia.net/CERT/Software/asax/
NetSTAT	University of California at Santa Barbara	http://www.cs.ucsb.edu/~kemm/netstat.html/
Prelude	Yoann Vandoorselaere voann@mandrakesoft.com Laurent Oudot oudot.laurent@wanadoo.fr	http://www.prelude-ids.org/
SHADOW	Naval Surface Warfare Center, Dahlgren Division	http://www.nswc.navy.mil/ISSEC/CID
Snort	Martin Roesch	http://www.snort.org/
SnortNet	Fyodor Yaroshkin	http://snortnet.scorpions.net/

Таблица 1.1. Некоммерческие системы обнаружения компьютерных атак.

Методика и критерии сравнения

Сравнительный анализ вышеназванных систем производился следующим образом: был составлен набор критериев сравнения качественных характеристик СОА, для которых определено множество возможных значений. Далее по каждому из критериев оценивалось соответствие ему каждой системы в отдельности и сравнение полученных характеристик систем в совокупности.

Для сравнительного анализа были выбраны следующие критерии:

Класс обнаруживаемых атак. Данный критерий определяет, какие классы атак способна обнаруживать рассматриваемая система. Это один из ключевых критериев, так как сегодня ни одна система не способна обнаруживать атаки всех классов. Поэтому для более полного покрытия всего спектра атак необходимо комбинировать различные СОА. Здесь мы используем классификацию атак, основанную на разделении ресурсов защищаемой системы по типам.

Класс атаки – это четверка <**L**,**R**,**A**,**D**>, где **L** – расположение атакующего объекта, **R** – атакуемый ресурс, **A** – целевое воздействие на ресурс, **D** – признак распределенного характера атаки.

L: расположение атакующего объекта. Оно может быть внутренним по отношению к защищаемой системе (***li***), либо внешним (***le***).

R: атакуемый ресурс. Ресурсы разделяются по расположению и по типу.

По расположению: локальные (***rl***), сетевые (***rn***).

По типу: пользовательские ресурсы (***ru***), системные ресурсы (***rs***), ресурсы СУБД (***rd***), вычислительные ресурсы (***rc***), ресурсы защиты (***rp***).

A: целевое воздействие на ресурс: сбор информации (***as***), получение прав пользователя ресурса (***au***), получение прав администратора ресурса (***ar***), нарушение целостности ресурса (***ai***), нарушение работоспособности ресурса (***ad***).

D: признак распределенного характера атаки: распределенные (***dd***), нераспределенные (***dn***).

Следующий критерий характеризует источники и способы сбора информации о поведении объектов и состоянии ресурсов:

Уровень наблюдения за системой. Определяет, на каком уровне защищаемой системы собирают данные для обнаружения атаки. Различаются системные и сетевые источники. В пределах системных источников разделяются уровни ядра, процесса и приложения. От уровня наблюдения за системой зависит скорость сбора информации,

влияние системы на собираемую информацию, вероятность получения искаженной информации.

В одном случае СОА может иметь собственные наблюдающие модули, которые в реальном времени получают информацию из первичных источников – из канала передачи данных в сети, системной шины узла сети, ядра ОС узла, от встроенных в приложения модулей.

В другом случае СОА может использовать лишь вторичные источники информации – журналы регистрации ОС узла, приложений, сетевые данные, собранные другими средствами (на других узлах). В этом случае вероятность получение искаженных данных выше и заведомо меньше скорость обнаружения возможной атаки.

Следующий критерий определяет эффективность обнаружения атаки на основе анализа полученной информации.

Используемый метод обнаружения. Метод обнаружения также является ключевым критерием сравнения. Методы подразделяются на формальные и эвристические. Формальные методы базируются на механизмах определения текущего состояния ресурсов системы и оценки опасности этих состояний. В этих методах ля каждого состояния одного ресурса должна быть однозначно определена мера «безопасности». Существующие системы используют эвристические методы обнаружения, которые обнаруживают и классифицируют атаки на основе неполной информации и с некоторой долей вероятности неправильного обнаружения и классификации.

Адаптивность к неизвестным атакам. Определяет, позволяет ли используемый метод обнаруживать ранее неизвестные атаки. Экспертные методы требуют постоянного обновления базы знаний об атаках описаниями новых атак, которые производятся экспертами. Пока не построено эффективного представления базы знаний об атаках, которое позволило бы получать описания возможных новых атак на основе описания известных атак. Методы обнаружения аномального поведения объектов позволяют обнаруживать неизвестные атаки, но они также имеют много слабых сторон – неустойчивость к малым изменениям поведения объектов, низкая точность классификации атаки.

Следующие три критерия определяют такие архитектурные особенности СОА как управление, распределенность,

Управление. Определяет организацию управления системой. Управление может быть централизованное, распределенное. Дополнительно может присутствовать возможность удаленного управления СОА. Сюда включаются задачи установки, настройки и администрирования системы. При полностью распределенном управлении необходимо управлять всеми компонентами СОА в отдельности. При полностью централизованном управлении все компоненты СОА могут управляться с одного узла.

Архитектура. Определяет распределенность архитектуры СОА. Архитектура может быть нераспределенной или распределенной. Данная характеристика определяет скорость реакции СОА на атаку. При нераспределенной архитектуре СОА может собирать и анализировать информацию о поведении объектов в некотором ограниченном сегменте защищаемой системы. Она не может, в общем случае, оценить безопасность состояния всей защищаемой системы. Таким образом, распределенная архитектура СОА позволяет расширить горизонт наблюдения за защищаемой системой, который определяет время оценки защищенности системы и обнаружения атаки.

Расширяемость. Определяет, насколько система является открытой для интеграции в нее компонентов сторонних разработчиков и для сопряжения ее с другими системами защиты информации. Это могут быть программные интерфейсы для встраивания дополнительных модулей, реализация стандартов взаимодействия сетевых компонентов.

Формирование ответной реакции на атаку. Определяет наличие в системе встроенных механизмов ответной реакции на атаку, кроме самого факта ее регистрации. Возможны такие способы реагирования как разрыв соединения с атакующим объектом, блокировка его на межсетевом экране, отслеживание пути проникновения атакующего объекта в защищаемую системы.

Защищенность. Определяет степень защищенности СОА от атак на ее компоненты, включая защиту передаваемой информации, устойчивость к частичному выходу компонентов из строя или их компрометации. Затрагиваются вопросы наличия уязвимостей в компонентах СОА, защищенности каналов передачи данных между ними, авторизации компонентов внутри СОА.

Данный обзор строился, по большей части, на основе изучения описаний систем и их документации. Поэтому не изучены такие критерии сравнения как точность и полнота систем обнаружения атак, так как их изучение требует проведения большой экспериментальной работы.

Результаты

В данном разделе приводятся результаты сравнения рассмотренных семи СОА. Системы сначала сравниваются отдельно по каждому из критериев, а в конце раздела приводится сводная таблица сравнения по всем критериям.

Класс обнаруживаемых атак

Все исследованные системы могут обнаруживать атаки нескольких классов. Поэтому для улучшения читаемости и сокращения объема текста вводятся понятия объединения, пересечения и вложения классов атак. Для обозначения объединения и пересечения классов атак будем использовать символы "$\cup$" и "$\cap$" соответственно.

Системы, рассмотренные в данной работе, предназначены для обнаружения атак разных классов. Часть систем ориентированы на обнаружения локальных атак и при этом используют для анализа такие источники как журналы регистрации приложений, ОС, журналы систем аудита (AAFID, ASAX). Другие системы обнаруживают только внешние (сетевые) атаки и используют для анализа информацию, получаемую из каналов передачи данных в сети (SHADOW, Snort, SnortNet). Остальные системы являются гибридными и обнаруживают как локальные атаки, так и внешние атаки (NetSTAT, Prelude).

В пределах атак тех классов, на которые они ориентированы, все системы имеют различную полноту обнаружения, т.е. потенциальное число обнаруживаемых атак (отношение числа обнаруженных атак к числу проведенных атак). Это зависит от используемого метода обнаружения и от полноты базы описаний известных атак (если она есть).

Система AAFID является наименее развитой в этом плане, так как все еще не вышла за рамки экспериментальной системы. Набор обнаруживаемых атак для нее довольно беден. Система имеет набор агентов, обнаруживающих аномалии для конкретных групп ресурсов. Она обнаруживает следующие аномалии:

- аномалии загрузки процессора;
- аномалии входа в систему;
- аномалии прав доступа к системным файлам;
- аномалии конфигурации NFS и др.

Система обнаруживает атаки следующих классов: (L,R,A,D). Где:

- L=$\{li\}$ (атакующие объекты внутренние для системы);
- R=$\{rl\} \cap \{ru \cup rs \cup rc\}$ (локальные пользовательские, системные ресурсы и вычислительные ресурсы);
- A=$\{as \cup au \cup ar \cup ad\}$ (сбор информации о системе, попытки получения прав пользователя, попытки получения прав администратора, нарушение работоспособности ресурса);
- D=$\{dn\}$ (нераспределенные).

Система ASAX единственная из рассмотренных здесь является изначально ориентированной на обнаружения атак уровня системы

(локальных). Она появилась раньше всех из рассмотренных здесь систем, и последняя версия предназначена, в частности, для анализа журналов регистрации системы аудита Solaris BSM. ASAX имеет набор правил на языке RUSSEL для обнаружения определенного набора атак для некоторых платформ. Эти правила включают в себя обнаружение несанкционированного доступа к системным файлам, сигнатуры известных атак на почтовые сервисы, обнаружение нарушения прав доступа к программам suid и пр. ASAX более эффективен, чем AAFID, так как долго использовался в целевых ОС и, соответственно, имеет более полную базу описаний известных атак.

Обнаруживаются атаки следующих классов: (L,R,A,D). Где:

- L={*li* } (атакующие объекты внутренние для системы);
- R={*rl*} ∩ {*ru* ∪ *rs*} (локальные пользовательские и системные ресурсы);
- A={*au* ∪ *ar* ∪ *ad*} (попытки получения прав пользователя, попытки получения прав администратора, нарушение работоспособности ресурса);
- D={*dn*} (нераспределенные).

Система NetSTAT также пока находится в стадии экспериментальной разработки. Для компонентов, анализирующих сетевые данные, есть описания некоторого набора известных атак на типовые сервисы. Локальные системные компоненты ориентированы на анализ журналов регистрации системы аудита Solaris BSM. Базы описаний известных атак малы, поэтому эффективность системы низка.

СОА обнаруживает атаки следующих классов: (L,R,A,D). Где:

- L={*li* ∪ *le*} (внутренние и внешние атаки);
- R={*rl* ∪ *rn* } ∩ {*ru* ∪ *rs*} (атаки на локальные или сетевые пользовательские ресурсы и системные ресурсы);
- A={*as* ∪ *au* ∪ *ar* ∪ *ad*} (сбор информации о системе, попытки получения прав пользователя, попытки получения прав администратора и нарушение работоспособности ресурса);
- D={*dn*} (нераспределенные).

Система Prelude, как и NetSTAT, является гибридной, т.е. способна обнаруживать атаки как на уровне системы, так и на уровне сети. Данная система активно разрабатывается в настоящее время. По классам обнаружения сетевых атак Prelude превосходит систему Snort, так как способна использовать базу описаний атак этой системы в дополнение к собственной. Системная часть Prelude имеет достаточно широкий набор описаний атак и использует в качестве источника информации различные журналы регистрации:

- журналы регистрации межсетевого экрана IPFW, входящего в состав ОС FreeBSD;

- журналы регистрации NetFilter ОС Linux 2.4.x;
- журналы регистрации маршрутизаторов Cisco and Zyxel;
- журналы регистрации GRSecurity;
- журналы регистрации типовых сервисов ОС UNIX.

СОА обнаруживает атаки следующих классов: (L,R,A,D). Где:

- L={*li* ∪ *le*} (внутренние и внешние атаки);
- R={*rl* ∪ *rn*} ∩ {*ru* ∪ *rs* ∪ *rp*} (атаки на локальные или сетевые пользовательские ресурсы, системные ресурсы и ресурсы защиты);
- A={*as* ∪ *au* ∪ *ar* ∪ *ad*} (сбор информации о системе, попытки получения прав пользователя, попытки получения прав администратора и нарушение работоспособности ресурса);
- D={*dn*} (нераспределенные).

Система SHADOW является сетевой системой обнаружения атак. Она представляет собой систему фильтров собираемых сетевых данных и обнаруживает простейшие аномалии в сети. При этом она является довольно требовательной к ресурсам и требует выделенного оборудования для каждого из своих компонентов.

Системой обнаруживаются атаки следующих классов (L,R,A,D):

- L={*li* ∪ *le*} (внутренние и внешние атаки);
- R={*rn*} ∩ {*ru* ∪ *rs*} (атаки на сетевые пользовательские ресурсы и системные ресурсы);
- A={*as* ∪ *au* ∪ *ar* ∪ *ad*} (сбор информации о системе, попытки получения прав пользователя, попытки получения прав администратора и нарушение работоспособности ресурса);
- D={*dn*} (нераспределенные).

Система Snort это наиболее популярная на сегодняшний день некоммерческая СОА. Она активно и динамично развивается, обновления базы известных атак происходят с частотой, сравнимой с коммерческими аналогами. Snort является чисто сетевой СОА и, кроме основной базы описаний атак, имеет набор подключаемых модулей для обнаружения специфических атак или реализующих альтернативные методы обнаружения.

Системой обнаруживаются атаки следующих классов (L,R,A,D):

- L="внутренние" ∪ "внешние";
- R={*rl* ∪ *rn*} ∩ {*ru* ∪ *rs* ∪ *rp*} (атаки на локальные или сетевые пользовательские ресурсы, системные ресурсы и ресурсы защиты);

- A={***as*** ∪ ***au*** ∪ ***ar*** ∪ ***ad***} (сбор информации о системе, попытки получения прав пользователя, попытки получения прав администратора и нарушение работоспособности ресурса);

- D={***dn***} (нераспределенные).

Система SnortNet это распределенная СОА, основанная на системе Snort. В SnortNet добавляется управляющая станция и набор агентов пересылки сообщений для организации распределенной сети сенсоров Snort. Классы обнаруживаемых атак у систем SnortNet и Snort эквивалентны.

Таким образом, по критерию обнаруживаемых классов атак, наиболее эффективными и перспективными представляются системы Prelude и Snort (SnortNet).

Уровень наблюдения за системой

Из рассмотренных семи систем все работают с данными уровня приложений на системном уровне и с сетевыми данными. То есть анализируемая информация получается из вторичных источников, таких как журналы регистрации приложений, ОС, либо из сетевого канала передачи данных.

Системы AAFID и ASAX работают только с журналами регистрации приложений и операционной системы. Системы SHADOW, Snort, SnortNet анализируют только сетевые. Системы NetSTAT и Prelude анализируют как данные из локальных системных источников, так и сетевые данные.

Используемый метод обнаружения

Все рассмотренные системы используют в качестве основного метода обнаружения атак экспертный метод.

Система AAFID имеет в своем составе набор агентов, которые в явном виде используют характеристики атак, заданные человеком, реализовавшим агенты. При этом не используется никакого формального языка описания атак. Агенты представляют собой программные модули, написанные на алгоритмическом языке общего назначения, в которых жестко определены признаки тех атак, для обнаружения которых они предназначены. Такая организация базы описаний известных атак является трудно расширяемой.

Система ASAX использует язык описания сценариев атак RUSSEL, который по описательной мощности эквивалентен алгоритмическому языку C [1]. Используется архитектурно-независимый формат представления данных журналов регистрации

NADF (Normalized Audit Trail Format). Добавление нового источника данных можно осуществить, добавив конвертер формата данных в NADF и добавив сценарии атак при необходимости.

Система NetSTAT использует язык описания сценариев атак STATL, особенностью которого является возможность описания сценария атаки в виде последовательности состояний атакуемого ресурса. Таким образом, эта система использует метод обнаружения, близкий к формальным методам.

Система Prelude использует отдельные базы сигнатур атак для сетевых данных и для журналов регистрации. Для анализа сетевых данных можно импортировать сигнатуры системы Snort. Также используется набор специализированных модулей для обнаружения специфических атак, таких как сканирование портов, некорректные ARP-пакеты. Специальные модули производят дефрагментацию IP, сборку TCP-потока, декодирование HTTP-запросов.

Система SHADOW использует набор фильтров на языке Perl - в сенсоры и анализаторы «зашиты» знания разработчиков системы обнаружения атак о том, какие сетевые пакеты могут быть признаками атаки. Подобно системе AAFID является трудно расширяемой.

Система Snort использует базу сигнатур известных атак. Также используется набор специализированных модулей для обнаружения специфических атак, таких как сканирование портов, отправка большого числа фрагментированных пакетов. Специальные модули производят дефрагментацию IP, декодирование HTTP-запросов. В 2001 году появился модуль статистического анализа потока сетевых данных, но его эффективность неизвестна.

Система SnortNet эквивалентна системе Snort по методам обнаружения.

По данному критерию наиболее перспективными представляются системы с открытой архитектурой, позволяющие встраивать альтернативные методы обнаружения атак. Это системы Prelude, Snort (SnortNet) и NetSTAT.

Адаптивность к неизвестным атакам

На данный момент эта возможность отсутствует у большинства рассмотренных СОА. Возможно использование экспериментального модуля статистического анализа системы Snort, но его эффективность не изучена. Таким образом, по данному критерию также справедлив предыдущий вывод, наиболее перспективны системы Prelude, Snort (SnortNet) и NetSTAT, которые позволяют встраивать альтернативные методы обнаружения атак.

Управление

Для некоммерческих систем характерна слабая проработка вопросов удобства пользования и администрирования. Поэтому большинство систем управляются локально с тех узлов, на которых установлены их компоненты. Некоторые системы имеют подсистемы управления удаленно через web-интерфейс (SHADOW, Prelude), но возможности этих интерфейсов ограничены.

Система AAFID управляется централизованно с основного монитора системы (пользовательского интерфейса). При этом основной монитор может управлять вторичными мониторами. Подробнее об архитектуре системы см. раздел 6.

Система ASAX управляется централизованно на том узле, где она установлена, при помощи файлов конфигурации.

Система NetSTAT управляется распределенно через файлы конфигурации на всех узлах, где расположены компоненты системы.

Система Prelude управляется централизованно при помощи управляющей консоли. Компоненты системы сами предоставляют управляющей консоли те параметры их функционирования, которые могут изменяться. Также управление может осуществляться через локальные конфигурационные файлы на тех узлах, где установлены компоненты COA.

Система SHADOW управляется распределенно через файлы конфигурации на всех узлах, где расположены компоненты системы.

Система Snort управляется централизованно через файлы конфигурации, консольные команды и сигналы UNIX.

Система SnortNet управляется распределенно через файлы конфигурации на всех узлах, где расположены сенсоры Snort. При этом используется централизованная станция мониторинга.

Архитектура

Системы ASAX и Snort являются нераспределенными, остальные системы имеют распределенную архитектуру. Детальное описание архитектуры каждой из систем приводится в разделе 6.

Расширяемость

Все рассмотренные системы, кроме SHADOW, являются расширяемыми за счет добавления новых модулей со

стандартизированным интерфейсом, а также за счет использования стандартных протоколов обмена сообщениями.

AAFID имеет открытый интерфейс для добавления новых агентов и фильтров.

ASAX имеет открытый интерфейс для добавления новых источников информации.

NetSTAT имеет открытый интерфейс для добавления новых агентов и фильтров.

Prelude имеет открытый интерфейс для добавления новых модулей анализа и реагирования, ведения журналов регистрации. Обмен сообщениями происходит по стандарту IDMEF (Intrusion Detection Message Exchange Format), оптимизированному для высокоскоростной обработки.

Snort имеет открытый интерфейс для добавления новых модулей анализа, имеется модуль, реализующий протокол SNMPv2.

SnortNet аналогична системе Snort, а также использует стандарт обмена сообщениями IAP (Internet Alert Protocol). В дальнейшем планируется реализовать обмен сообщениями в формате IDMEF.

Наиболее расширяемыми являются системы Prelude и Snort.

Формирование ответной реакции на атаку

Встроенную возможность реагирования на атаку имеют системы NetSTAT, Prelude и Snort. В системе NetSTAT это реализовано лишь в тестовом варианте. Система Prelude имеет набор агентов ответной реакции, которые могут блокировать атакующего при помощи межсетевого экрана. Ведутся работы по агентам, способным изолировать атакующего, либо уменьшить пропускную способность его канала. Система Snort имеет встроенную ограниченную возможность реагирования на атаку путем отправки TCP-пакетов, разрывающих соединение (с установленным флагом RST), а также ICMP-пакетов, сообщающих атакующему узлу о недоступности узла, сети или сервиса.

Защищенность

Все системы, которые пересылают какие-либо данные, используют для этого защищенные каналы. AAFID неустойчива к возможным атакам, направленным на нее саму, в силу особенностей реализации – использование алгоритмического языка Perl, который позволяет изменение программного кода на этапе выполнения. NetSTAT, Prelude, SnortNet используют библиотеку OpenSSL для шифрования канала между компонентами. SHADOW использует

протокол SSH. Snort реализует протокол SNMPv2, в котором присутствуют функции шифрования паролей при передаче данных.

СОА Prelude имеет дополнительные механизмы, обеспечивающие безопасность ее компонентов. В системе используется специализированная библиотека, которая делает безопасными такие библиотечные функции алгоритмического языка С функции как printf, strcpy, которые не проверяют размер передаваемых им данных. Библиотека предотвращает классические ошибки выхода за границы массивов и переполнения буферов.

Дополнительные модули анализа сетевых данных делают систему устойчивой к некорректным сетевым пакетам на разных уровнях стека и выходу ее компонентов из строя. Такие атаки как отправка пакетов с неправильными контрольными суммами, обнуленными флагами TCP, ресинхронизация сессий, случайная отправка и «обрезание» сегментов системой игнорируются.

Из рассмотренных систем вопрос безопасности наиболее проработан в системе Prelude.

3.10. Итоговая таблица по критериям сравнения

Ниже в табл. 3.1 приведены сводные результаты сравнения рассмотренных систем по выбранным критериям.

	AAFID	ASAX	NetSTAT	Prelude	SHADOW	Snort	SnortNet
Классы атак	({*li*}, {rl} ∩ {ru ∪rs ∪ rc}, {*as* ∪ *au* ∪ *ar* ∪ *ad*}, {*dn*})	({*li* }, {*rl*} ∩ {*ru* ∪ *rs*}, {*au* ∪ *ar* ∪ *ad*}, {*dn*})	({*li* ∪ *le*}, {*rl* ∪ *rn)* ∩ (*ru* ∪ *rs*}, {*as* ∪ *au* ∪ *ar* ∪ *ad*}, {*dn*})	({*li* ∪ *le*}, {*rl* ∪ *rn)* ∩ {*ru* ∪ *rs* ∪*rp*}, {*as* ∪ *au* ∪ *ar* ∪ *ad*}, {*dn*})	({*li* ∪ *le*}, {*rl* ∪ *rn)* ∩ {*ru* ∪ *rs*}, {*as* ∪ *au* ∪ *ar* ∪ *ad*}, {*dn*})	({*li* ∪ *le*}, {*rl* ∪ *rn)* ∩ {*ru* ∪*rs*∪*rp*}, {*as* ∪ *au* ∪ *ar* ∪ *ad*}, {*dn*})	({*li* ∪ *le*}, {*rl* ∪ *rn)* ∩ {*ru* ∪*rs*∪*rp*}, {*as* ∪ *au* ∪ *ar* ∪ *ad*}, {*dn*})
Уровень наблюдения за системой	системный	системный	системный, сетевой	системный и сетевой	сетевой	сетевой	Сетевой
Метод обнаружения	экспертный	экспертный	экспертный, контроль переходов состояний	экспертный	экспертный	экспертный, статисти-ческий анализ	Экспертный, статисти-ческий анализ
Адаптивность	-	-	-	-	-	+	+
Управление	Централизо-ванное	централизо-ванное	распреде-ленное	централизо-ванное	распреде-ленное	централизо-ванное	распреде-ленное
Архитектура	Распреде-ленная	нераспреде-ленная	распреде-ленная	распреде-ленная	распреде-ленная	нераспреде-ленная	распреде-ленная
Расширяемость	Програм-мный интерфейс	програм-мный интерфейс	програм-мный интерфейс	програм-мный интерфейс, IDMEF	-	програм-мный интерфейс, SNMPv2	програм-мный интерфейс, IAP
Реакция	-	-	+	+	-	+	+
Защита	-	-	SSL	SSL, Libsafe	SSH	-	SSL

Таблица 3.1. Результаты сравнения СОА.

Описание исследованных систем

В данном разделе приводится более детальное описание рассмотренных в данной работе СОА. Для каждой системы описывается ее архитектура, используемая платформа и некоторые индивидуальные особенности.

AAFID (Autonomous Agents for Intrusion Detection)

Система AAFID разработана в университете Purdue, , West Lafayette, IN, USA. Первые публикации по системе датированы 1998 г., последние – 1999 г. AAFID – это одновременно название распределенной архитектуры систем обнаружения атак и собственно системы обнаружения атак [3,5,6,16]. Основой системы являются *автономные агенты* обнаружения. Наиболее интересна в системе именно ее архитектура.

Основными компонентами системы являются: *агенты*, *фильтры*, *трансиверы*, *мониторы*.

Агент наблюдает за конкретными аспектами функционирования узла и уведомляет соответствующий трансивер об аномальных событиях на узле. Агенты не могут непосредственно генерировать сообщения для пользователя. Таким образом, трансиверы получают полную информацию о функционировании узла, а мониторы – о сети в целом.

Фильтры предназначены для сбора однотипной информации из различных источников (различных системных журналов или от наблюдателей). Так же фильтры выполняют функции уровня представления модели OSI (Open System Interconnect): преобразовывают форматы данных, собираемых из аналогичных источников, но на разных архитектурах, к единому виду. Один фильтр может предоставлять данные нескольким агентам. На каждый источник данных существует только один фильтр. Агенты могут *подписываться* на получение данных от фильтра.

Трансивер – интерфейс взаимодействия между узловыми компонентами системы обнаружения и сетевыми компонентами системы обнаружения. Основными функциями трансивера являются управление агентами на данном узле (запуск, останов), сбор и анализ данных от агентов, передача данных и результатов анализа мониторам или другим агентам.

Мониторы – наивысшие в иерархии компоненты системы AAFID. Функционально они похожи на трансиверы с тем отличием, что сбор и анализ информации происходит не на одном узле, а на части сети

или на всей сети. Мониторы могут управлять трансиверами и другими мониторами. Мониторы имеют механизмы взаимодействия с пользовательским интерфейсом и являются точками доступа к системе обнаружения атак.

Система все еще находится на стадии прототипа. Последнее обновление имело место в сентябре 1999 года и данная версия была реализована и протестирована под операционными системами Solaris и Linux. Языком реализации является алгоритмический язык perl. На нем реализованы все компоненты системы.

ASAX (Advanced Security audit trail Analyzer on uniX)

Система ASAX разработана в университете Namur, Belgium. Первые публикации по системе датированы 1991 г., последние – 1998 г. Система предназначена для анализа журналов регистрации ОС UNIX [1,4,8,12].

Первоначально система ASAX была разработана под практически несовместимые ОС BS2000 и SINIX компании Siemens-Nixdorf A.G. Она представляла собой систему анализа журналов регистрации, для которых тогда не существовало единого стандарта. Поэтому задача построения системы разбивалась на две большие подзадачи: разработать унифицированный и гибкий формат журнальных записей, в которые можно было бы корректно преобразовывать исходные журналы регистрации ОС, разработать эффективный, мощный и при этом удобный язык описания и обнаружения сложных шаблонных конструкций в журнальных записях. Первую задачу решала сама компания Siemens-Nixdorf A.G., решением второй задачи занимался Университет Namur в Бельгии.

В результате исследовательской работы был разработан язык RUSSEL, в качестве модели анализа потоков данных вообще и анализа журналов безопасности в частности. В качестве единого формата журнальных записей был разработан т.н. *нормальный формат журнальных записей* (NADF – Normalized Audit Data Format). Единственное требование к формату являлась возможность прямого преобразования записей из специфических форматов ОС в NADF.

Язык RUSSEL предназначен для описания правил обработки записей в формате NADF. Каждое правило состоит из двух частей – описание *условия* срабатывания правила и *действия*. Действием могут быть вывод сообщения, вызов другого правила. Система обнаружения состоит из интерпретатора языка RUSSEL и источников информации, преобразующих записи системных журналов в записи в формате NADF.

Система ASAX реализована и существует только для платформ UNIX и реализована на алгоритмическом языке C.

NetSTAT (Network-based State Transition Analysis Tool)

Система NetSTAT является развитием проекта Калифорнийского Университета в г.Санта-Барбара STAT – средство анализа состояний переходов (State Transition Analysis Tool) [7,13,17,18]. Первые публикации по системе датированы 1992 г., последние – 2001 г. Основой используемого системой метода является описание исходной защищаемой системы в виде набора состояний ее компонентов и последующий анализ переходов из состояния в состояние в результате активных внешних воздействий.

Состояния защищаемой системы определяются при настройке и конфигурации системы обнаружения атак. Для каждого состояния определяется характеристика защищенности. Определяются *переходы* – изменения состояния защищаемой системы. В результате, атаки описываются в виде последовательностей переходов. Описание атак в виде последовательности переходов терминах состояний призвано избежать традиционных ограничений методов, основанных на сигнатурах и описывать шаблоны для целых классов типовых атак.

Основой системы является язык STATL – расширяемый язык, предназначенный для описания шаблонов атак в терминах STAT. Базовый язык оперирует наиболее абстрактными понятиями, не зависящими от конкретной системы и ее конфигурации. Язык позволяет достраивать себя, добавляя специфичные для конкретной системы *события*. Для каждого нового события описывается его *предикат*. Все конструкции языка STATL и его расширения транслируются в язык C++.

В системе могут использоваться *модули реакции* – связанные с ядром компоненты, осуществляющие реагирование на обнаруженную атаку.

Языком реализации системы NetSTAT является язык C++. Она работает под управлением ОС Linux и Solaris.

SHADOW (Secondary Heuristic Analysis for Defensive Online Warfare)

Система SHADOW разработана в Naval Surface Warfare Center, подразделении морской пехоты США. Система изначально разрабатывалась в открытых исходных текстах [14].

SHADOW работает под управлением ОС Linux, реализована на языке perl и состоит из двух типов компонентов – сенсора сетевого трафика и анализатора. Они могут находиться на одном узле, или на разных узлах.

Сенсор основан на библиотеке libpcap и утилите tcpdump, которые широко доступны на различных платформах UNIX. Требованием для сенсора является наличие выделенной машины для его работы. Задачей сенсора – сбор и архивирование сетевого трафика по типам протоколов (задается административно при конфигурировании сенсора).

Анализатор собирает данные с сенсоров и применяет к ним специализированные фильтры, содержащие сигнатуры известных атак и сигнатуры, заданные пользователем.

Сенсоры и анализаторы должны располагаться на выделенных быстродействующих компьютерах. Наибольшие требования производительности предъявляются компьютеру, на котором работает анализатор – им должна быть машина с максимально возможным быстродействием центрального процессора (производительностью узлов определяется порог скорости передачи данных в каналах связи, до превышения которого не будет пропущен ни один сетевой пакет).

Prelude

Система Prelude начала разрабатываться в 1998 год. Она изначально задумывалась как гибридная СОА, которая могла бы помочь администратору сети отслеживать активность как на уровне сети, так и на уровне отдельных узлов. Система распределенная и состоит из следующих компонентов [20]:

сетевые сенсоры – различные сенсоры, анализирующие данные на уровне сети на основе экспертного анализа. Сенсоры генерируют сообщения об обнаружении аномалий и отправляют их модулям управления. Система Prelude имеет собственную базу сигнатур атак, а также может использовать сигнатуры в формате системы Snort, что позволяет эффективно обновлять базу сигнатур атак;

системные сенсоры – различные сенсоры уровня системы, анализирующие журналы регистрации ОС, приложений. Сенсоры генерируют сообщения об обнаружении аномалий и отправляют их модулям управления. Существующий набор сенсоров позволяет анализировать данные журналов регистрации таких систем и приложений как межсетевй экран IPFW, входящий в состав ОС FreeBSD, NetFilter ОС Linux 2.4.x, маршрутизаторы Cisco и Zyxel, GRSecurity; типовые сервисы ОС UNIX.;

модули управления – процессы, которые получают и обрабатывают сообщения сенсоров. Различаются следующие виды модулей управления:

- модули журнализации – отвечают за регистрацию сообщений в журналах регистрации или базах данных. В настоящее время реализованы модули для хранения сообщений в базах данных MySQL, PostgreSQL;
- модули реагирования – анализируют сообщение и генерируют возможную ответную реакцию СОА на атаку. Возможны такие виды реакции как блокирование нарушителя на межсетевом экране (NetFilter, IPFilter). В дальнейшем возможны такие типы реакции как изоляция нарушителя и сужение пропускной способности канала нарушителя.

агенты реагирования – реализуют сгенерированную менеджером реакцию на атаку;

интерфейс – основан на протоколе http. Предоставляет возможность получать статистику и управлять системой при помощи web-браузера.

У системы Prelude есть несколько особенностей, которые отличают ее от других современных некоммерческих СОА. Система везде, где возможно, построена на использовании открытых стандартов. Так, для обмена сообщениями используется формат IDMEF (Intrusion Detection Message Exchange Format), оптимизированный для высокоскоростной обработки. Это позволяет в дальнейшем интегрировать компоненты в системы сторонних производителей и наоборот.

При разработке системы особое внимание было уделено вопросам безопасности. Каналы передачи данных шифруются по протоколу SSL, кроме того, используется специализированная библиотека, которая предотвращает классические ошибки выхода за границы массивов и переполнения буферов.

Дополнительные модули анализа сетевых данных делают систему устойчивой к некорректным сетевым пакетам на разных уровнях стека и выходу ее компонентов из строя. Такие атаки как отправка пакетов с неправильными контрольными суммами, обнуленными флагами TCP, ресинхронизация сессий, случайная отправка и «обрезание» сегментов системой игнорируются и они не приводят к отказу компонентов СОА.

Snort

Система Snort является классическим продуктом с открытыми исходными текстами [15].

Основным методом обнаружения атак, используемым в системе, является обнаружение злоупотреблений на основе описания сигнатур атак. В системе используется простой язык описания сигнатур атак, который полностью описан в документации и позволяет администраторам системы дополнять базу сигнатур своими сигнатурами. Каждое правило на этом языке состоит из двух частей: условие применения и действие.

В настоящее время система находится в стадии активной разработки, и каждые несколько месяцев появляются новые версии системы и новые функции.

Архитектура системы Snort целиком разрабатывалась из соображений эффективности и скорости работы. Поэтому она предельно проста и состоит из трех подсистем: декодер пакетов, ядро обнаружения и подсистемы оповещения и реагирования. Декодер пакетов реализует набор процедур для последовательной декомпозиции пакетов в соответствии с уровнями сетевого стека, то есть принятый кадр последовательно преобразуется в пакет, сегмент и блок данных с применением специфичных для данного уровня сигнатур атак.

Кроме модуля анализа трафика на основе правил, к ядру обнаружения могут подключаться модули сторонних разработчиков и производить анализ на одном из уровней декомпозиции пакета. С помощью таких модулей можно добавлять функциональности ядру обнаружения атак и реализовывать различные методы обнаружения. В последних версиях появился модуль статистического анализа, который предназначен для обнаружения аномалий в сетевом трафике.

Подсистема оповещения и реагирования отвечает за сохранение результатов анализа трафика в журналы регистрации самой системы Snort, либо вывод этой информации через системные службы регистрации событий ОС. Например, в UNIX-подобной ОС это может быть сервис регистрации событий *syslog*.

Система Snort имеет реализации под множество UNIX платформ, а так же под ОС компании Microsoft.

SnortNet

SnortNet это распределенное расширение системы Snort, целью которого является придание ей дополнительных возможностей по масшабируемости и расширяемости [21].

Система состоит из нескольких программных модулей: сенсоров, модулей пересылки сообщений и станции мониторинга. Система позволяет осуществлять мониторинг сетевого трафика и осуществлять информирование станции мониторинга обо всех обнаруженных аномалиях в поведении сетевых объектов. Система

использует Snort в качестве сетевого сенсора. В качестве протокола обмена данными система использует протокол Internet Alert Protocol (Протокол передачи сигналов тревоги – IAP). Для шифрования каналов передачи данных, аутентификации и контроля доступа система использует библиотеки SSL и TCP wrappers.

Обнаружение атак производится сенсорами Snort , после чего сообщения отправляются по протоколу IAP через модули пересылки сообщений (proxy) на станцию мониторинга. Модули пересылки сообщений и станцию мониторинга рекомендуется располагать в демилитаризованной зоне сети.

Система SnortNet протестирована под операционными системами Linux, FreeBSD, OpenBSD.

Заключение

Роль систем обнаружения атак для обеспечения информационной безопасности неуклонно возрастает. Стоимость систем и затраты на их эксплуатацию для коммерческих СОА очень велика. Учитывая рост популярности открытого программного обеспечения и его существенно более низкую, по сравнению с коммерческим программным обеспечением, стоимость, представляется, что роль некоммерческих СОА будет расти еще быстрее.

Рассмотренные в данной работе СОА сильно отличаются друг от друга по качественным характеристикам. Сравнение количественных характеристик может послужить предметом отдельного исследования.

По ключевым критериям сравнения систем можно сделать следующие выводы:

Классы обнаруживаемых атак. Часть из рассмотренных систем ориентированы на обнаружение исключительно сетевых атак, часть – на обнаружение системных атак. Две системы являются гибридными. Поэтому при выборе СОА для внедрения в систему можно либо выбирать несколько наиболее эффективных специализированных сетевых и системных СОА и устанавливать их в дополнение друг к другу, либо брать одну гибридную систему. Первый вариант может обеспечить высокую начальную эффективность обнаружения атак, но при этом дороже в поддержке, так как придется одновременно поддерживать и внедрять две или более несовместимых или слабо совместимых системы. Второй вариант может иметь меньшую начальную эффективность, но будет дешевле в эксплуатации. Таким образом, использование гибридных СОА представляется более предпочтительным. Наиболее эффективны в этом смысле системы Prelude, Snort, SnortNet.

Методы обнаружения. Наиболее распространенные сегодня методы обнаружения атак не позволяют системе адаптироваться к новым атакам и обнаружение атак начинается лишь после явного внесения их описания в базу знаний системы. Такие системы требуют постоянной поддержки. Более сложные методы пока используются не очень широко, но их роль неизбежно будет возрастать.

Архитектура. Распределенные СОА сложнее в установке и администрировании, но они сильно выигрывают у нераспределенных СОА в масштабируемости и расширяемости. Для крупных сетей распределенные СОА более предпочтительны.

Открытая архитектура необходима также для сопряжения СОА с другими средствами защиты информации в сетях. Одним из слабых мест коммерческих СОА является поддержка стандартов в области обнаружения атак. Открытая архитектура и поддержка открытых стандартов позволит некоммерческим системам занять достойное место на рынке. Наиболее эффективнор проработана архитектура в системах AAFID, NetSTAT, Prelude.

Защита. Собственная защита СОА часто является довольно слабой. Некоторые коммерческие СОА требуют выделения отдельной технологической сети для связи компонентов. Устойчивость СОА к атакам является одним из ключевых вопросов обеспечения безопасности в сети, частью системы защиты которой является эта СОА.

Из рассмотренных 7 систем наиболее полно удовлетворяют всем критериям следующие 3 системы: Prelude, Snort, SnortNet. При этом система Prelude менее популярна, но имеет несколько важных преимуществ перед системой Snort – распределенность, масштабируемость и расширяемость.

Литература

1. Mounji, Languages and Tools for Rule-Based Distributed Intrusion Detection, PhD Thesis, Computer Science Institute, University of Namur, Belgium, Sept 1997.

2. Amoroso, Edward, G., Intrusion Detection, 1st ed., Intrusion.Net Books, Sparta, New Jersey, USA, 1999.

3. Jai Balasubramaniyan, Jose Omar Garcia-Fernandez, E. H. Spafford, Diego Zamboni, An Architecture for Intrusion Detection using Autonomous Agents, Department of Computer Sciences, Purdue University; Coast TR 98-05; 1998.

4. Baur & W. Weiss, "Audit Analysis Tool for Systems with High Demands Regarding Security and Access Control", Research Report, ZFE F2 SOF 42, Siemens Nixdorf Software, München, November 1988.

5. M. Crosbie, E. Spafford, Defending a computer system using autonomous agents, Technical Report 95-022, COAST Laboratory, Department of Computer Sciences, Purdue University, West Lafayette, IN 47907-1398, March 1994.

6. M. Crosbie, E. Spafford, Defending a computer system using autonomous agents, in: Proceedings of the 18th National Information Systems Security Conference, October 1995.

7. S.T. Eckmann, G. Vigna, and R.A. Kemmerer, "STATL: An Attack Language for State-based Intrusion Detection," in Proceedings of the ACM Workshop on Intrusion Detection, Athens, Greece, 2000.

8. N. Habra, B. Le Charlier, A. Mounji & I. Mathieu, "Preliminary Report on Advanced Security Audit Trail Analysis on UniX", Research Report 1/92, Institut d'Informatique, University of Namur, January 1992.

9. N. Habra, B. Le Charlier, A. Mounji, Advanced Security Audit Trail Analysis on uniX. Implementation Design of the NADF Evaluator. Research Report, Computer Science Institute, University of Namur, Belgium, March 1993.

10. S. Kumar, Classification and detection of computer intrusions, Ph.D. Thesis, Purdue University, West Lafayette, IN 47907, 1995.

11. Kumar, S. & Spafford, E. (1995) A Software Architecture to Support Misuse Intrusion Detection. Department of Computer Sciences, Purdue University; CSD-TR-95-009.

12. Mounji, B. Le Charlier, D. Zampuniéris, N. Habra, Preliminary Report on Distributed ASAX. Research Report, Computer Science Institute, University of Namur, Belgium, May 1994

13. Porras, P. A., Ilgun, K., and Kemmerer, R. A. (1995). State transition analysis: A rule-based intrusion detection approach. IEEE Transactions on Software Engineering,SE-21: 181–199.

14. Ralph, William D., SHADOW version 1.7 Installation manual, 2001, http://www.nswc.navy.mil/ISSEC/CID/Install.pdf

15. Roesch, Martin, Snort Users Manual, Snort Release: 1.8.1, 2001, http://www.snort.org/

16. *Eugene H. Spafford, Diego Zamboni,* Intrusion detection using autonomous agents, Computer Networks, 34(4):547-570, October 2000.

17. G. Vigna, R. Kemmerer, "NetSTAT: A Network-based Intrusion Detection Approach," in Proceedings of the 14th Annual Computer Security Application Conference, Scottsdale, Arizona, December 1998.

18. G. Vigna, R.A. Kemmerer, "NetSTAT: A Network-based Intrusion Detection System," Journal of Computer Security, 7(1), IOS Press, 1999.

19. Diego Zamboni , E. H. Spafford, *AAFID2 Users Guide.* Department of Computer Sciences; 1998.

20. Yoann Vandoorselaere, Laurent Oudot, "Prelude, an Hybrid Open Source Intrusion Detection System", http://www.prelude-ids.org/, 2002.

21. Yaroshkin Fyodor, "SnortNet – A Ditributed Intrusion Detection System", IVT-1/95, Kyrgyz Russian Slavic University, Bishkek, Kyrgystan, June 2000.

Раздел III
Машинная графика и обработка образов

Подшивалов А.Ю.

Эффективные алгоритмы анализа взаимного расположения сильно разветвленных пространственных объектов

Работа посвящена проблеме создания алгоритмов анализа взаимного расположения объектов, имеющих большие размеры и протяженность вдоль нескольких произвольно ориентированных осей (орбитальные станции и их сегменты). Спецификой алгоритмов является, с одной стороны, - необходимость учета ресурсов бортовых компьютеров, и, с другой стороны, - необходимость анализировать в реальном времени трехмерные сцены, содержащие сотни тысяч и миллионы примитивов. Кроме этого выдвигаются требования к высокой точности вычислений в связи с высоким риском столкновений объектов. В работе рассмотрены "жесткие" объекты (узлы сетки, описывающей поверхность, взаимно неподвижны), а также деформируемые объекты (включая объекты с подвижными "жесткими" конструкциями и объекты с произвольными деформациями). Рассматриваются различные виды деформаций, для каждого из которых представлены эффективные алгоритмы анализа столкновений.

Введение

Проблема анализа взаимного положения тел различной формы изучается уже достаточно длительное время, и существует широкий круг алгоритмов, посвященных решению этой проблемы. Все алгоритмы можно разделить на две категории: универсальные, не зависящие от способа задания объекта, и специальные, разработанные для конкретных моделей (подробнее см. [1]).

В данной работе речь идет об универсальных алгоритмах, предназначенных для анализа объектов, представленных в виде набора выпуклых многоугольников (полигонов).

Вообще говоря, для проведения анализа взаиморасположения объектов сцены необходимо рассмотреть все пары участвующих в ней объектов. Для уменьшения числа пар, требующих проверки,

используются различные методы, в основу которых, как правило, положены два подхода - разбиение сцены на непересекающиеся области и использование ограничивающих объемов.

Разбиение пространства сцены позволяет делать заключение о том, что пара объектов не пересекается, если объекты принадлежат разным областям. На этой идее, например, основан метод двоичного разделения пространства, BSP – Binary Space Partitioning [2]. Чем больше расстояние между объектами сцены, тем проще удается выполнить их разделение. Метод BSP широко применяется в алгоритмах визуализации, однако очевидно, что при малых расстояниях между объектами сложной формы разделение пространства весьма затруднительно - удается отделить лишь части объектов, особенно если объекты невыпуклые, разветвленные и т.п.

К классу методов, основанных на разделении пространства, относятся также "воксельные" методы. Среди наиболее эффективных – разбиение пространства по принципу октантного дерева. Узлы октантного дерева можно представить как шаги иерархического процесса разбиения пространства тремя взаимно ортогональными плоскостями так, что каждый воксель-родитель содержит восемь вокселей-потомков [5]. На этапе предварительных вычислений для каждого объекта осуществляется построение октантного дерева заданной глубины. Данные, которые содержит каждый узел дерева позволяют определить, принадлежит ли данный воксель объекту. Основное достоинство таких деревьев состоит в том, что над объектами, погруженными в октантные деревья, очень эффективно выполняются логические операции, в том числе логическое "и" (пересечение). Кроме того, произвольная деформация объекта сводится лишь к изменению значений узлов октантного дерева. Основным недостатком октантных деревьев (и воксельных методов вообще) является большой объем данных, необходимых для хранения информации о деревьях, поскольку зависимость числа узлов дерева от линейных размеров объекта является кубической.

Другой класс алгоритмов основан на использовании разного рода ограничивающих объемов. Объект или его часть заключаются внутрь ограничивающего объема, как правило, достаточно простой формы. Если ограничивающие объемы объектов не пересекаются, то и сами объекты не имеют общих точек. Наиболее распространенным типом объемов являются сферы и параллелепипеды, стороны которых параллельны осям координат, так называемые AABB – Axis Aligned Bound Box [6]. Их основным преимуществом является простота проверки на перекрывание. Недостаток AABB состоит в том, что возможны большие зазоры между объектом и охватывающим его

объемом. Величина зазора для AABB также зависит от начальной ориентации объекта.

Большинство алгоритмов, рассмотренных выше, быстро отбрасывают пары непересекающихся объектов в сценах, где расстояние между объектами сравнимо с характерными размерами самих объектов. Особенностью сцен, рассматриваемых в нашем случае, является возможность плотных контактов и соприкосновений между объектами сложной формы. Использование рассмотренных выше алгоритмов в таких сценах приводит к резкому увеличению проверок на перекрывание, включая проверки даже на уровне отдельных узлов моделей объектов.

Наиболее перспективным в нашем случае видится метод с использованием деревьев на основе ориентированных ограничивающих объемов (Oriented Bound Boxes - OBB), предложенный в [3]. Основные мотивы выбора данного подхода в качестве базового для построения алгоритма, предлагаемого автором, следующие:

- каждый из OBB представляет собой параллелепипед с собственной ориентацией в пространстве (это позволяет получить плотное облегание объекта независимо от его начального расположения);
- OBB-дерево имеет гораздо меньшую сложность, чем сферическими деревья или деревья AABB (это важно при использовании бортовых вычислителей);
- в [3] предлагается быстрый алгоритм проверки двух OBB на пересечение (это важно для объектов, сложных по форме и с высокой пространственной разветвленностью).

Метод ориентированных ограничивающих объемов (Oriented Bound Boxes - OBB)

Логически весь метод можно разделить на несколько составляющих:

- построение OBB-дерева;
- проверка двух OBB на перекрывание;
- проверка двух примитивов (треугольников) на перекрывание;
- проверка пары объектов на пересечение.

Рассмотрим вкратце каждую из составляющих.

Построение OBB дерева. При построении OBB-дерева выполняется два типа операций: построение объема (облегающего объект или часть объекта) и построение иерархии ограничивающих объемов. Будем считать, что оболочка, облегающая объект или его часть, образована сеткой из n треугольников. Для OBB,

ограничивающего эту сетку, требуется определить ориентацию и длину вдоль каждой из его пространственных осей.

Для нахождения ориентации OBB согласно [3] необходимо вычислить общий центр масс m и ковариационную матрицу C,. Пусть p^i, q^i, r^i – вершины i-го треугольника (векторы из трех элементов). Тогда:

$$m = \frac{1}{3n}\sum_{i=0}^{n}(p^i + q^i + r^i)$$

$$C_{jk} = \frac{1}{3n}\sum_{i=0}^{n}(\overline{p}_j^i \overline{p}_k^i + \overline{q}_j^i \overline{q}_k^i + \overline{r}_j^i \overline{r}_k^i), \quad 1 \le i,k \le 3$$

где C_{jk} – элементы ковариационной матрицы **C**, а векторы

$$\overline{p}_j^i = p^i - m, \quad \overline{q}_j^i = q^i - m, \quad \overline{r}_j^i = r^i - m,$$

Таким образом, получена симметрическая ковариационная матрица C размером *3x3*. В качестве ортонормированного базиса, задающего оси OBB в пространстве, будем использовать нормированые собственные векторы полученной матрицы.

Для определения размеров OBB используется понятие *экстремальных вершин*. Вершина является экстремальной по данной координате, если среди рассматриваемого множества вершин эта координата имеет экстремальное значение. Размером OOB считается размер параллелепипеда, на гранях которого лежат экстремальные вершины.

Процесс последовательного дробления охватывающего объект объема можно представить иерархической структурой, в которой корнем дерева является объем, охватывающий весь рассматриваемый объект. Для последующих дроблений объема применяется следующее правило. Упорядочим оси OBB по убыванию длины OBB вдоль каждой из них. Деление объема происходит вдоль самой длинной из осей с помощью плоскости, ортогональной самой оси. Треугольники, принадлежащие объему, делятся между объемами-потомками в зависисимости от того, по какую сторону от разделяющей плоскости лежит центр треугольника. Двухмерный аналог такого деления показан на рис.1. Координата деления вдоль оси определяется положением центра масс m. Этот факт основывается на предположении построения сбалансированного дерева. Если не удается разделить объем вдоль самой длинной оси, выбирается следующая по длине ось и повторяются те же действия. Случай, когда деление не может быть выполнено ни в одном направлении, определяет окончание процесса. Таким образом, каждому объему-листу дерева может принадлежать один или несколько треугольников.

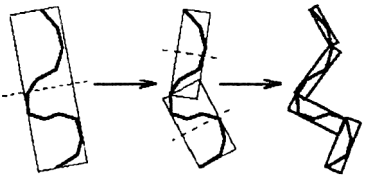


Рис.1. Построение OBB дерева. Двумерная проекция.

Сложность построения OBB дерева оценивается как ***O(n lgn)***. Дествительно, для построения каждого слоя дерева требуется ***O(n)*** операций, а глубина дерева определяется как ***O(lg n)***. Важно учесть, что построение OBB-деревьев объектов выполняется на предварительном этапе вычислений.

Проверка двух OBB на перекрывание. В процессе работы алгоритма анализа взаиморасположения объектов основное время занимает тестирование пар OBB на перекрывание. Поскольку стандартный способ проверки (проверить все пары граней) занимает слишком много времени, предлагается использовать алгоритм, описанный в [3], как наиболее эффективный из известных автору.

Выберем некоторую прямую в пространстве и спроектируем на нее рассматриваемые OBB. Каждая проекция представляет собой отрезок этой прямой. Если отрезки не пересекаются, то не пересекаются и сами OBB. В этом случае выбранная прямая называется "разделяющей прямой", а плоскость, перпендикулярная этой прямой, "разделяющей плоскостью". В противном случае необходим дальнейший анализ.

Определим множество прямых, которые необходимо проверить. По теореме о разделяющей плоскости [4] два многоугольника, рассматриваемых в 3D пространстве, не пересекаются тогда и только тогда, когда существует разделяющая их плоскость, параллельная двум прямым, каждая из которых проходит через ребро одного из многоугольников.

Каждый OBB имеет три направляющих вектора. Пары ребер, оба из которых принадлежащат одному OBB, дают три плоскости. Пары ребер, каждое из которых принадлежит различным OBB, дают еще 9 плоскостей. В итоге получаем 15 плоскостей. Если хотя бы одна из плоскостей является разделяющей, то OBB не пересекаются, в противном случае у OBB существуют общие точки. Таким образом, для того, чтобы узнать, пересекаются ли два OBB, необходимо проверить 15 плоскостей на разделение.

Рассмотрим подробнее алгоритм проверки каждой плоскости. Нормаль к этой плоскости и есть возможная разделяющая прямая. Спроецируем на нее центры обоих OBB и вычислим длины отрезков,

являющихся проекциями каждого ОВВ на эту прямую. Заметим, что центр ОВВ проецируется на середину отрезка. Поэтому, если расстояние между проекциями центров будет больше суммы полудлин отрезков, то отрезки не пересекаются, а значит прямая является разделяющей (рис. 2).

Пусть даны два ОВВ, ***A*** и ***B***. Пусть в системе координат ***A*** объем ***B*** задается матрицей поворота ***R*** и вектором трансляции (сдвига) ***T***. Пусть полудлины ребер ***A*** и ***B*** равны соответственно a_1, a_2, a_3, и b_1, b_2, b_3, а направления ребер ОВВ задаются как векторы A^i и B^i, где $i = 1, 2, 3$ (т.е. всего 6 векторов).

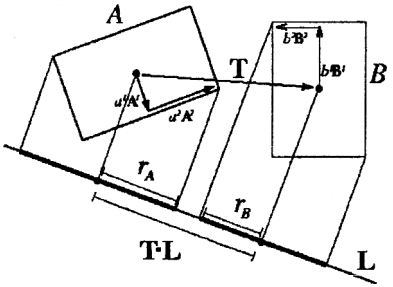


Рис. 2. Проверка прямой на разделение двух ОВВ
A, ***B*** – рассматриваемые ОВВ, ***L*** – разделяющая прямая

Найдем длины каждого полуотрезка r_a и r_b. Поскольку векторы направлений ОВВ взаимно ортогональны, то длина проекции ОВВ на прямую равна сумме длин проекций каждого из трех ребер, т.е.

$$r_a = \sum_{i=1}^{3} \left| a_i A_i \cdot L \right|$$

Расстояние между проекциями центров ОВВ равно $|T \cdot L|$. Отсюда получаем искомое неравенство:

$$\left| T \cdot L \right| > \sum_{i=1}^{3} \left| a_i A_i \cdot L \right| + \sum_{i=1}^{3} \left| b_i B_i \cdot L \right|$$

Если оно выполняется, то интервалы, а значит, и ОВВ, не пересекаются. Заметим, что неравенство упрощается в случае, когда прямая параллельна одному из ребер, или векторному произведению пары ребер.

Пусть, например, $L = A^1 \times B^2$. В этом случае, второй член первой суммы преобразуется следующим образом:

$$\left|a_2 A^2 \cdot \left(A^1 \times B^2\right)\right| = \left|a_2 B^2 \cdot \left(A^2 \times A^1\right)\right| = \left|a_2 B^2 \cdot A^3\right| = \left|a_2 B_3^2\right| = a_2 \left|R_{32}\right|$$

Последнее равенство в этой формуле получено исходя из того, что мы рассматриваем матрицу $\boldsymbol{R}$ как матрицу поворота объема $\boldsymbol{B}$ в системе координат объема $\boldsymbol{A}$. В этом случае $\boldsymbol{3}$ вектора $\boldsymbol{B^i}$ совпадают со столбцами матрицы $\boldsymbol{R}$.

Упростив аналогично исходное неравенство для остальных членов первой и второй сумм, после сложения и приведения подобных (часть членов успешно сокращается) мы получаем окончательное неравенство в виде:

$$\left|T_3 R_{22} - T_2 R_{32}\right| = a_2 \left|R_{32}\right| + a_3 \left|R_{22}\right| + b_1 \left|R_{13}\right| + b_3 \left|R_{11}\right|$$

Все 15 плоскостей могут быть проверены подобным образом. В худшем случае, когда выполняются все 15 тестов, мы получаем результат в 200 операций сложения и умножения. В среднем число операций будет существенно меньше.

Примечательно, что данный алгоритм (с минимальными изменениями) успешно работает даже для вырожденных OBB (прямоугольник, либо отрезок) и для OBB, имеющих сторону бесконечной длины.

Проверка двух треугольников на перекрывание. В случае, когда объемы нижнего уровня пересекаются, необходимо проверять на пересечение сами треугольники. Алгоритм проверки в общих чертах таков.

Сначала отсеиваются непересекающиеся треугольники. Для это требуются минимальные затраты, если в качестве разделяющих прямых поочередно использовать координатные оси (выбор максимума и минимума из трех чисел и выполнение двух операций сравнения). В ситуации, когда отсев не происходит, отыскивается разделяющая прямая аналогично тому, как это делается для пары OBB. В данном случае количество проверяемых прямых сокращается до 11.

Проверка пары объектов на пересечение. Проверка производится, начиная с корневых OBB, охватывающих каждый объект целиком, и далее вплоть до листьев дерева. Критерием отсутствия пересечения объектов (подобъектов) является отсутствие пересечения OBB. При пересечении OBB на данном уровне проводится проверка для пар OBB следующего уровня иерархии. Если один из OBB является листом дерева, то процедура проверки вызывается для него и двух OBB-потомков второго дерева. Если же оба OBB являются листьями, то процедура проверки вызывается для всех пар треугольников, принадлежащих соответствующим OBB.

Следует отметить, что наличие пересечения двух OBB лишь сужает область поиска. Наличие же пересечения треугольников

указывает на наличие контакта между объектами и точно определяет узлы, в которых сеточные модели объектов пересекаются.

Особенности использования метода OBB для 3D-объектов высокой сложности.

При моделировании поведения объектов сложной пространственной конфигурации (например, космических станций и их сегментов) мы имеем дело с трехмерными моделями, содержащими существенно невыпуклые поверхности (такие поверхости можно сравнить с веткой «обычного» дерева). Здесь большое значение имеет тот факт, что если процесс дробления облегающего объема для такого объекта строить по схеме с единым деревом иерархии, глубина дерева будет слишком большой, что в свою очередь приведет к резкому увеличению расхода памяти и потере эффективности.

Для предотвращения подобного развития процесса разбиения автором предлагается выделять сложные по форме участки объекта в отдельные подобъекты. Для каждого из подобъектов будем строить свое OBB-дерево. Таким образом, мы получим объект с «не слишком невыпуклой» поверхностью, содержащий список подобъектов. Несмотря на увеличение количества используемых в сцене OBB-деревьев, проверка на пересечение данного объекта с другим будет требовать меньшее количество OBB-тестов. Понятно, что могут наблюдаться случаи, когда подмены ветвей на отдельные объекты не принесут результата (как раз для объектов, похожих на «обычное» дерево). Однако, для рамок данной работы эти случаи можно отнести к исключительным. Более характерным примером является, в частности, орбитальная станция с большим количеством выступающих над ее корпусом конструкций (солнечные батареи, стрелы манипуляторов, антенны и т.п.).

Другим аспектом процесса моделирования поведения сложных объектов является огромное количество (сотни тысяч) узлов моделей объектов, в связи с чем для бортовых компьютеров возникает проблема емкости оперативной памяти. Основная нагрузка при этом ложится на хранение моделей объектов и их OBB-деревьев. Например, если построить сбалансированное дерево для объекта, состоящего из n примитивов, оно будет иметь глубину lgn и содержать $2n-1$ узлов (т.е. даже в два раза больше, чем число самих узлов модели).

Предлагаемое автором решение проблемы состоит в выборе компромисса между производительностью алгоритмов и затратами памяти. Возможности здесь достаточно велики. Например, при уменьшении глубины дерева на 3, число его узлов (а значит и расход памяти) уменьшается уже в 8 раз.

В некоторых случаях, по-видимому, может быть постановлена задача оптимизации, где (в зависимости от ситуации) в качестве критерия и ограничений могут выступать либо затраты памяти, либо быстродействие.

Анализ столкновений для деформируемых объектов

На сегодняший день среди публикаций, посвященных проблеме анализа столкновений автору встречались работы, в которых рассматриваются только "жесткие" объекты, т.е. объекты, узлы сетки которых неподвижны относительно друг друга. Вместе с тем представление моделей поверхностей объектов в виде статичных сеток не позволяет эффективно решать целый ряд важных задач анализа сложных 3D-сцен в реальном времени. В этой связи автором предлагается рассмотреть более общую постановку задачи анализа столкновений, в которой сетка, моделирующая поверхность объекта, может существенно меняться в течение характерного времени решения задачи (для обеспечения работы системы в режиме реального времени необходимо выполнять по крайней мере 20-25 итераций в секунду).

Введем понятие деформации сетки. Будем называть деформацией любые изменения координат узлов сетки, описывающей объект, а также изменения связей между ними. Под понятием связь в данном случае подразумевается треугольник, соединяющий узлы сеточной модели.

Автором предлагается рассмотреть несколько классов для описания относительно простых видов деформациями сетки, которые можно считать своего рода базисом для описания объекта с произвольной деформацией:

- объекты с подвижными жесткими частями;
- объекты с предопределенным множеством состояний;
- объекты с подвижными узлами сетки.

Объект с произвольной деформацией можно представить с помощью комбинации указанных классов объектов. Анализ взаиморасположения для всех классов деформаций опирается на рассмотренный выше метод OBB. Отличие состоит в дополнительных структурах данных и действиях, которые необходимы для описания деформаций.

Коротко рассмотрим каждый из упомянутых классов.

Объекты с подвижными жесткими частями. Сеточные модели таких объектов имеют подвижные участки, тем не менее внутри каждого такого участка объект остается «жестким». Простым примером подобного объекта является модель человеческой руки:

плечо $\Rightarrow$ предплечье $\Rightarrow$ кисть $\Rightarrow$ палец и т.д.

Выделим каждый такой участок в отдельный объект со своим собственным OBB-деревом, а из модели исходного объекта удалим узлы сетки, принадлежащие подвижному участку. Объект-родитель содержит список своих объектов-потомков. Объект-потомок, в свою очередь, также может быть составным. Таким образом мы приходим к иерархическому представлению объектов с подвижными жесткими частями. При анализе взаиморасположения составных объектов необходимо также проверять каждую пару подобъектов внутри дерева иерархии.

Объекты с предопределенным множеством состояний. Данный класс описывает объекты, для которых можно заранее определить все его возможные *состояния*, т.е. набор узлов сеточной модели и построенное на ее основе OBB-дерево. Например, переключатель можно рассматривать как объект с двумя состояниями: включен и выключен. Зададим для объекта *активное состояние*, т.е. состояние, в котором он находится в текущий момент. Таким образом, объект однозначно задается множеством своих состояний и указанием активного состояния, а суть деформации заключается в смене активного состояния.

На этапе подготовки объектов к вычислениям в оперативную память загружаются все состояния объекта. Это означает, что для каждого из них вычисляется OBB дерево, и, вместе с сеточной моделью объекта, сохраняется в памяти. При анализе взаиморасположения объектов используется OBB-дерево, соответствующее активному состоянию объекта. Из соображений затрат памяти на практике нет смысла использовать больше двух-трех состояний.

Объекты с подвижными узлами сетки. Данному классу принадлежат объекты, для которых заранее предсказать деформацию невозможно. Суть деформации здесь заключается в изменении координат узлов сеточной модели объекта. Проблема состоит в том, что некоторые узлы сетки могут выйти за пределы ограничивающих их объемов. Поэтому необходима корректировка OBB-дерева в режиме реального времени. Автором предлагается следующий простой алгоритм для приведени OBB-дерева в соответствие с новой сеткой:

Для каждого измененного узла выберем соответствующий OBB нижнего уровня.

Проверяем принадлежность узла OBB. Если узел не принадлежит OBB, изменяем границы OBB так, чтобы охватить узел.

Если OBB является корнем дерева, конец. Иначе, переходим к OBB-родителю и повторяем пункт 2.

Оценим сложность преобразования дерева. Пусть мы изменили координаты m узлов сетки. Поскольку глубина дерева оценивается как

O(lgn), нам необходимо выполнить ***O (m lgn)*** проверок OBB с возможной коррекцией.

Отметим, что многократное использование смещений большого количества узлов сетки может в значительной мере уменьшить самое главное свойство OBB дерева – плотное облегание объекта. Если планируется использовать такие объекты, следует ограничиться AABB деревьями.

Представление объектов с произвольной деформацией. Произвольную деформацию объектов будем рассматривать в виде совокупности рассмотренных выше классов деформации по следующим правилам. Прежде всего выделим подвижные подобъекты и построим иерархическое дерево объектов. Затем определим состояния объектов, для которых это возможно сделать. К этому же классу относятся объекты, для которых предполагается изменение связей между узлами сетки. Объекты, возможная деформация которых неизвестна заранее, будем рассматривать на уровне объектов с подвижными узлами.

Таким образом, анализ взаиморасположения для деформируемых объектов может быть выполнен на основе метода OBB с минимальными потерями эффективности использования оперативной памяти и времени вычислений.

Заключение

В данной работе рассмотрена проблема анализа взаиморасположения объектов сложной пространственной структуры. Основная задача работы: определить, пересекаются ли объекты, и если да, то в каких именно точках. К разрабатываемым алгоритмам были предъявлены следующие требования: эффективность, точность, возможность работы в системе с ограниченными вычислительными ресурсами (бортовой комплекс ЭВМ). В работе большое внимание уделяется деформируемым объектам с произвольной деформацией.

Автором разработаны высокоэффективные алгоритмы анализа взаиморасположения. В основу положен метод ориентированных ограничивающих объемов (Oriented Bound Box), предложенный в [3]. В главе 2 рассмотрены проблемы построения OBB-дерева для сильноразвлетвленных пространственных объектов в условиях с ограниченными ресурсами памяти. В главе 3 рассматриваются деформируемые объекты. Поставлена задача анализа взаиморасположения произвольно деформируемых объектов, представленных с помощью сеточной модели. Автором сделана попытка выделить отдельные классы деформации, для которых эффективно решается поставленная задача, а также показано, как с

помощью совокупности различных классов можно представить и исследовать произвольно деформируемые объекты.

Применение предложенных алгоритмов рассчитано на системы моделирования реального времени, в первую очередь, при моделировании поведения космических станций.

Литература

1. R.O.Duda and P.E.Hart. *Pattern classification and scene analysis.* John Wiley and Sons, 1973.
2. Ming C. Lin, Stefan Gottschalk. *Collision detection between geometric models: a survey.* University of North Carolina.
3. S.Gottschalk, M.C.Lin and D.Manocha. *OBBTree: A hierarchical structure for rapid interference detection.*
4. S. Gottschalk. *Separating axes theorem.* Technical report TR96-024, Department of Computer Science, UNC Chapel Hill, 1996.
5. Y. Yang and N. Thalmann. *An improved algorithm for collision detection in cloth animation with human body.* First Pacific Conference on Computer Graphics and Application, pp. 237-251, 1993.
6. D. J. Cohen, M.C. Lin, D. Manocha, and M. Ponamgi. *I-Collide: An interactive and exact collision detection system for large-scale environments.* Proceeding of Symposium of Interactive 3D Graphics, pp. 189-196, 1995.

Буряк Д.Ю., Визильтер Ю.В.

Метод автоматизированного конструирования процедур обнаружения объектов на цифровых изображениях

Данная статья посвящена проблеме автоматизированного конструирования процедур анализа изображений. В работе приведено описание созданного метода для автоматизированного построения близкой к оптимальной процедуры идентификации объекта на изображении по его эталонному изображению с использованием изображений обучающей выборки. Метод построения искомой процедуры заключается в ее поиске среди некоторого подмножества процедур обнаружения заданного объекта, которое формируется по эталонному изображению и с использованием алгоритмов заданных типов. В качестве базового алгоритма была выбрана процедура согласованной фильтрации. Для реализации указанного поиска применялись генетические алгоритмы. Частный случай данного метода был реализован в виде программной системы и протестирован. Результаты экспериментов обсуждаются в заключительной части статьи.

Введение

В настоящее время накоплен большой опыт в решении задач распознавания изображений. Предварительный анализ данного опыта позволяет выделить ряд характерных особенностей, а именно:

- Разработано большое число алгоритмов и подходов к решению различных задач. Однако для подавляющего числа этих алгоритмов нельзя заранее предсказать, насколько оправдано (в смысле быстродействия, точности обнаружения и т.п.) их применение к конкретной проблеме.
- Практически для каждой задачи всегда можно подобрать несколько алгоритмов, решающих ее, но выбор из них наиболее подходящего по некоторому набору критериев в большинстве случаев основывается на эвристических принципах и результатах тестирования.

- В большинстве случаев для решения конкретных задач применяются различные комбинации уже известных алгоритмов, поскольку разработка совершенно нового алгоритма решения требует колоссального опыта работы в области компьютерного зрения.
- Несмотря на быстрый рост производительности вычислительной техники актуальной остается проблема получения оптимального решающего алгоритма, который удовлетворяет заданной точности и затрачивает минимальное время.

Таким образом, средства, позволяющие автоматизировать процессы выбора и настройки алгоритмического обеспечения для получения оптимальной решающей процедуры, должны существенно облегчить практическое использование методов анализа изображений.

Исходя из вышесказанного, проблема разработки подходов к автоматизированному построению оптимальных процедур решения конкретной задачи распознавания изображений является актуальной и требует исследований.

Анализ состояния вопроса

В настоящее время существует несколько подходов к разработке систем построения и выбора процедур для решения задач машинного зрения.

Первый подход основывается на использовании библиотек процедур, реализующих различные алгоритмы обработки и обнаружения объектов на изображении. Обычно такие библиотеки содержат несколько сотен процедур, которые разделяются на группы: геометрические преобразования, фильтрация, обнаружение элементарных объектов (прямых, окружностей и т.п.) и т.д. Часто вместе с библиотеками пользователю предоставляется графическая оболочка, обеспечивающая возможность опробовать предлагаемые процедуры на конкретных изображениях. К программным системам, реализующим первый подход, относятся система *Aphelion* компании *ADCIS*, группа средств *eVision* компании *Euresys*, система *KBVision* компании *Amerinex Applied Imaging*, библиотеки *Tina*, которые разрабатываются компаниями *Artificial Intellegence Vision Research Unit* и *Electronic Systems Group* совместно с *университетом Манчестера* и *Neuro-Imaging Analisys Centre*.

В основе второго подхода лежит идея визуального программирования. Системы, построенные по этому принципу, также содержат библиотеки процедур обработки изображений, но кроме этого они позволяют пользователю спроектировать требуемую систему,

применяя приемы визуального программирования. Подобные программные системы не требуют глубокого знания языков программирования. К системам, реализующим второй подход, относятся: семейство систем *Khoros* компании *Khoral Research*, система *NeatVision* компании *NeatVision*.

К недостаткам систем, построенных на основе первых двух подходов, следует отнести техническую сложность автоматического построения процедур обнаружения заданного объекта, поэтому для нахождения оптимального решения пользователь вынужден самостоятельно создавать и тестировать различные комбинации существующих в системе алгоритмов, решающих поставленную задачу обнаружения.

Третий подход связан с системами автоматического построения процедур анализа изображений. Подобные системы автоматически конструируют нужный алгоритм в соответствии с заданным формальным описанием задачи (например, по структурному описанию объекта, который должен быть найден на изображении) и требованиям к процедуре решения. В качестве примера подобной системы можно привести систему *PIP* (*Prolog Image Processing*), разработанную в Великобритании в *University of Wales Cardiff*. Входными данными для этой системы является структурное описание искомого объекта. Применяя к нему процедуру логического вывода, основанную на Прологе, получим алгоритм обнаружения объекта на изображении. Недостатком подобных систем является сложность составления формального описания задачи, обычно представляемого в виде набора спецификаций, особенно при работе с реальными изображениями.

Предлагаемый метод, который может быть отнесен к третьему из указанных подходов, осуществляет автоматизированное построение близкой к оптимальной процедуры распознавания объекта на изображении для конкретной задачи. Но в отличие от существующих систем он лишен указанного выше недостатка, т.к. для его применения не требуется составления спецификаций для формального описания задачи.

Постановка задачи

Метод оперирует бинарными изображениями связанных объектов. Исходными данными для него являются:

- эталонное изображение объекта поиска: Im;
- обучающая выборка изображений, для каждого из которых отмечено, содержит ли оно заданный объект: множество пар $\{(S_i, c_i)\}$, $i=1,2,\ldots,n$, где S_i - бинарное изображение, $c_i=1$, если S_i содержит Im;

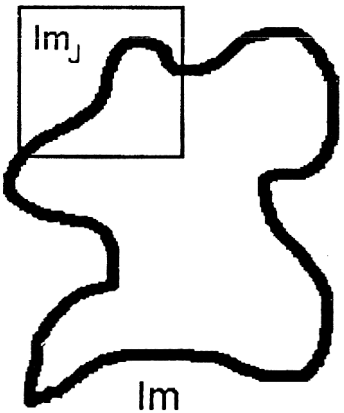
- пороговое значение по точности обнаружения: a.

Задачей данного метода является построение близкой к оптимальной процедуры обнаружения заданного объекта, которая затрачивает минимальное время и удовлетворяет заданной точности. Построенная процедура реализует алгоритмы заданных типов.

Метод построения искомой процедуры заключается в ее поиске среди некоторого подмножества процедур обнаружения заданного объекта, которое формируется по эталонному изображению и с использованием алгоритмов заданных типов.

Рассмотрим способ конструирования такого подмножества процедур.

На эталонном изображении Im случайным образом выберем m прямоугольных фрагментов $Im_j(x_j,y_j,w_j,h_j)$, где x_j,y_j - координаты центра фрагмента, w_j,h_j - его ширина и высота, соответственно, при этом x_j,y_j,w_j,h_j - являются значениями раномернораспределенных случайных величин (рис. 1).



Сопоставим каждому полученному таким образом фрагменту Im_j элементарную процедуру его обнаружения p_j. Данная процедура реализует один из заданных алгоритмов, хранящихся в базе знаний системы. Выбор базового алгоритма осуществляется случайным образом.

Следовательно, процедура $p=(p_1,p_2,...,p_m)$, которая заключается в последовательном применении построенных элементарных процедур, выполняет обнаружение множества объектов S', причем $Im \in S'$.

Пусть множество P содержит все возможные процедуры p^k, сконструированные указанным образом, т.е.

$$P = \{p^k \mid p^k = (p_1^k, p_2^k, ..., p_{m_k}^k)\},$$

где p_j^k - является процедурой обнаружения некоторого фрагмента эталонного изображения и реализует один из заданных базовых алгоритмов ($j=1,2,...,m_k$; $k=1,2,3...$).

Степень оптимальности (в указанном выше смысле) каждой процедуры обнаружения из множества P будем определять по результатам ее применения к изображениям из обучающей выборки. Близкой к оптимальной будем называть оптимальную процедуру из ограниченного множества P.

Следовательно, исходная задача построения близкой к оптимальной процедуры обнаружения сводится к следующей задаче условной оптимизации:

$$p_{opt} = \arg\left(\min_{\substack{prec(p^k, S_i) > a \\ i=1,2,...n \\ \forall p^k \in P}} T(p^k)\right),$$

где: $T(p^k)$ - время работы процедуры p^k на изображениях из обучающей выборки; $prec(p^k, S_i)$ - функция вычисления точности обнаружения процедуры p^k на изображении S_i из обучающей выборки.

Поставленная задача условной оптимизации, вообще говоря, определена на неограниченном множестве.

Под длиной процедуры обнаружения будем понимать количество элементарных процедур, входящих в ее состав. Предположим, что искомая оптимальная процедура находится среди процедур длины не больше заданной, т.е.:

$p_{opt} \in P^l$,

P^l - множество процедур длины не больше l,

$$P^l = \{p^k \mid p^k = (p_1^k, p_2^k, ..., p_{m_k}^k), m_k \le l\}$$

Тогда получим следующую задачу условной оптимизации на ограниченном множестве:

$$p_{opt} = \arg(\min_{\substack{prec(p^k, S_i) > a \\ i=1,2,\ldots n \\ \forall p^k \in P^l}} T(p^k))$$

Выделим несколько особенностей поставленной задачи условной оптимизации.

- Пространство поиска является достаточно большим даже при небольшой длине l процедур обнаружения.
- Исследование свойств функции $T(p^k)$, в общем случае, является трудновыполнимым.
- Опираясь на опыт исследователей в решении задач анализа изображений, можно предположить, что функция $T(p^k)$ при указанных ограничениях в общем случае не является непрерывной, гладкой и унимодальной.

Таким образом, для решения данной задачи условной оптимизации целесообразно использовать генетические алгоритмы.

Предлагается следующая схема применения генетического алгоритма.

1. Каждому гену соответствует одна из элементарных процедур.
2. Хромосома - последовательность генов ограниченной длины. Каждая хромосома соответствует одной из процедур обнаружения, принадлежащих множеству P^l.

3. Функцию качества для хромосомы будем вычислять по значениям времени работы и точности обнаружения соответствующей процедуры на обучающей выборке. Рассмотрим общий вид функции качества:

$$F(p) = \sum_{i=1}^{n} f(p, S_i),$$

$$f(p, S_i) = t(p, S_i) + A * pen(p, S_i),$$

где p - процедура обнаружения заданного объекта; S_i - изображение из обучающей выборки; $t(p, S_i)$ - время работы процедуры p на изображении S_i; A - константа; $pen(p, S_i)$ - штрафная функция, принимающая значение 0, если процедура p удовлетворяет заданной

точности обнаружения a на изображении S_i; и равная $t(p,S_i)+\alpha(p,S_i,a)$ в противном случае.

Варьируя значением константы A, пользователь имеет возможность ослаблять или усиливать влияние ограничений по точности на результирующую процедуру обнаружения.

1. Операция скрещивания позволяет конструировать новые процедуры обнаружения на базе уже построенных. Новая процедура формируется путем перегруппировки составных частей (групп элементарных процедур) существующих решений (процедур обнаружения).

2. Операция мутации позволяет изменить параметры (x_j,y_j,w_j,h_j) для выбранной элементарной процедуры. Данные параметры определяют фрагмент эталонного изображения, для обнаружения которого используется выбранная элементарная процедура. Таким образом, изменяя фрагмент эталонного изображения, операция скрещивания позволяет настроить новую элементарную процедуру обнаружения.

3. Будем искать решение с наименьшим значением функции качества.

Интерпретация результата

Результатом применения описанного метода является последовательность элементарных процедур обнаружения фрагментов эталонного изображения, которая формирует искомую процедуру обнаружения. При этом данная последовательность удовлетворяет заданной точности и затрачивает минимальное время при обработке изображений из обучающей выборки. Отсюда следует два способа интерпретации получаемого результата.

1. Процедурная интерпретация. Поскольку теория генетических алгоритмов не гарантирует нахождения оптимального решения задачи оптимизации, то полученный результат понимается как близкая к оптимальной процедура обнаружения заданного объекта.

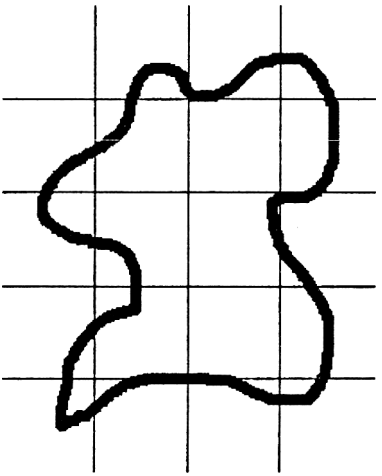
2. Объектная интерпретация. Каждая элементарная процедура, входящая в состав полученного решения, однозначно соответствует некоторому фрагменту эталонного изображения. Указанные фрагменты являются характерными фрагментами эталонного объекта для изображений из обучающей выборки. Кроме этого, т.к. последовательность найденных процедур удовлетворяет условию оптимальности, размеры каждого характерного элемента являются минимально возможными для его использования при обнаружении объекта поиска. Отсюда сформулируем объектную интерпретацию:

полученное решение определяет набор элементов минимального размера, с большой вероятностью идентифицирующих искомый объект на изображениях из обучающей выборки.

Реализация.

В виде программной системы был реализован частный случай описанного метода. Рассмотрим основные особенности и ограничения, принятые в данном варианте, по сравнению с общим случаем.

1. Для определения фрагментов искомого объекта, используемых при построении элементарных процедур, на эталонном изображении введена равномерная сетка (рис 2.), интервалы которой определяются пользователем. Все, полученные таким образом, фрагменты нумеруются.



2. В качестве базового алгоритма был принят алгоритм согласованной фильтрации. В этом случае точность обнаружения элементарной процедуры вычисляется как максимальное число совпадающих точек, деленное на размер окна фильтрации. Время работы элементарной процедуры выражается в количестве совершаемых ее операций сравнения и сложения.

3. Использована каноническая модель генетического алгоритма. Кодирование особи осуществляется через задание номера фрагмента эталонного изображения, используемого для определения

соответствующей элементарной процедуры обнаружения. Длина решающих процедур, в соответствии с канонической моделью, была зафиксирована и равнялась 4. Критерием окончания работы генетического алгоритма является количество совершенных итераций.

Входными данными разработанной программной системы являются:

- эталонное изображение объекта поиска;
- текстовый файл, содержащий список изображений обучающей выборки, для каждого изображения из которой указано, содержит ли оно заданный объект;
- пороговые значения по точности обнаружения;
- значение константы, используемой в функции оценки качества процедуры обнаружения;
- параметры сетки для определения фрагментов эталонного изображения;
- параметры генетического алгоритма, включающие вероятности скрещивания и мутации а также размер популяции и количество итераций.
- Система оперирует файлами в формате BMP.
- **Выходными данными системы являются:** фрагменты эталонного изображения, по которым построено найденное оптимальное решение.

Результаты тестирования

Данная программная система была протестирована на задаче идентификации дорожных знаков, имеющих достаточно сложную структуру и большое разнообразие.

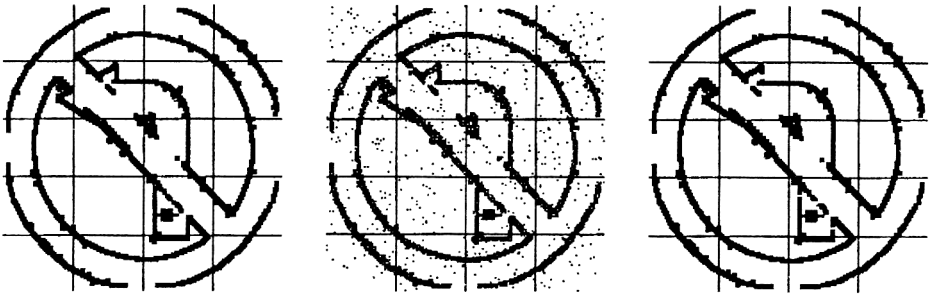
Для экспериментов были выбраны 14 различных дорожных знаков. Выбранные цветные изображения были предварительно обработаны для получения бинарного контурного препарата. На полученные бинарные изображения был наложен гауссовский шум. Обучающую выборку составили 14 "чистых" изображений и 14 зашумленных. Все изображения одинакового размера, и объекты располагаются в центре изображений.

Эксперименты проводились на задаче построения близкой к оптимальной процедуры идентификации знака "поворот налево запрещен". В качестве эталонного было выбрано незашумленное изображение с произведенными вручную искажениями. Часть этих

искажения была введена в незашумленном и зашумленном вариантах. Целью данных искажений было увеличить число характерных элементов данного знака.

Все эксперименты проводились при разбиении эталонного изображения решеткой 4*5.

На рисунке 3 представлен пример незашумленного, зашумленного и эталонного изображения знака "поворот налево запрещен" с наложенной сеткой.



Над указанными исходными данными были проведены эксперименты по решению следующих задач:

1. Найти процедуру идентификации заданного знака, демонстрирующую минимальную точность на изображениях обучающей выборки.
2. Найти процедуру идентификации заданного знака, демонстрирующую точность не хуже заданной и затрачивающую минимальное время на изображениях обучающей выборки.

Для проведения корректного сравнения точностно-временных характеристик получаемых решений предварительно, перед тестированием системы на каждой из данных задач, в заданном классе модульных алгоритмов, использующих согласованную фильтрацию, путем полного перебора были найдены оптимальные решающие процедуры. Перебор осуществлялся среди всех возможных модульных процедур заданной длины при заданном разбиении эталонного изображения. Отметим, что при длине решающей процедуры равной 4 и разбиении эталона сеткой 4*5, существует 20^4=160000 процедур обнаружения.

Рассмотрим результаты, продемонстрированные системой при решении первой из указанных задач. На рисунке 4 показаны фрагменты,

по которым осуществляется формирование оптимальной процедуры обнаружения, полученной после применения полного перебора.

При решении данной задачи при помощи разработанной программной системы использовались следующие значения параметров генетического алгоритма:

- Вероятность мутации: 0.1, 0.3, 0.5, 0.7;
- Вероятность скрещивания: 0.9;
- Размер популяции: 30, 50;
- Количество итераций: 100, 200, 300, 500.

Во всех решениях, полученных при применении мутации с вероятностью большей чем 0.1, присутствовали три фрагмента, определяющих оптимальную процедуру идентификации.

Рассмотрим результаты, полученные при решении второй из указанных задач. Отметим, что в данных экспериментах константе A, участвующей в функции качества, было присвоено значение 1. Таким образом, условия на точность не являются жесткими, т.е. точность получаемых процедур обнаружения находится в некоторой окрестности заданной точности. В качестве пороговых значений по точности были выбраны показатели, соответствующие оптимальной процедуре, найденной при решении предыдущей задачи. Для параметров генетического алгоритма были использованы значения, указанные выше.

На рисунке 5 изображены фрагменты, по которым построена оптимальная процедура, найденная путем полного перебора.

Данные фрагменты присутствовали в большинстве решений, полученных при использовании разработанной системы. Необходимо отметить, что помимо указанных фрагментов решения также содержали

элементы, определяющие оптимальную решающую процедуру, полученную при решении первой задачи.

Таким образом, данная программная система, реализующая описанный метод, позволяет находить близкие к оптимальным процедуры идентификации заданного объекта, как по критерию точности, так и по совокупности критериев точности и времени работы. При этом количество совершаемых операций в процессе поиска решения существенно меньше чем, например, при полном переборе.

Заключение

В заключение сформулируем основные полученные результаты.

- Разработан метод автоматизированного конструирования близких к оптимальным процедур идентификации объекта на изображении по его эталонному изображению и обучающей выборке.
- Осуществлена программная реализация частного случая предложенного метода.
- По результатам проведенных экспериментов решения задачи идентификации дорожных знаков было установлено, что данный метод действительно позволяет конструировать близкие к оптимальным процедуры обнаружения в заданном классе алгоритмов.

Работа выполнялась на кафедре АСВК факультета ВМиК МГУ им. М.В. Ломоносова в рамках проекта студенческой лаборатории Интел-МГУ, а также при поддержке гранта РФФИ 02-07-90130.

Автор благодарит своих научных руководителей Королева Л.Н. и Попову Н.Н. за ценные замечания и внимательное отношение к статье.

Литература

1. Дуда Р., Харт П. Распознавание образов и анализ сцен. - М.: Мир, 1976.

2. Курейчик В.М. Генетические алгоритмы. Обзор и состояние. Сборник "Новости искусственного интеллекта", №3 - 1998.

3. Ту Дж., Гонсалес Р. Принципы распознавания образов. - М.: Мир, 1978.

4. Banzhaf W., Nordin P., Keller R.E., Francone F.D. Genetic Programming - an Introduction: On the Automatic Evolution of Computer Programs and Its Applications. Dpunkt.verlag and Morgan Kaufmann Publishers, Inc. USA. 1998.

Зинченко Е.Ю., Попов А.М.

Исследование потери точности алгоритмов автоматической идентификации спикера по записи его речи

В работе проводится исследование широко распространенного подхода к задаче идентификации спикера в условиях помех различной природы. Подход основан на использовании квантования векторов признаков в пространстве кепстральных коэффициентов с последующей кластеризацией данных. Выбранная система идентификации показывает стопроцентное распознавание спикера в хорошо подобранных лабораторных условиях экспериментов. Показано, однако, что в условиях помех различной природы происходит потеря точности идентификации. Рассмотрены естественные помехи, ухудшающие качество микрофонной записи, а также активные помехи, связанные с присутствием других спикеров при записи.

Введение

Проблема автоматического распознавания параметров источника по косвенным измерениям сигналов является одной из фундаментальных проблем информатики, которой посвящено множество работ с начала появления вычислительных машин. Математические формулировки возникающих здесь проблем приводят к решению обратных задач той или иной степени сложности. Уровень, точность и амбициозность решаемых задач во многом определяется степенью развития вычислительной техники и возможностями регистрации сигналов. Интенсивное развитие вычислительной техники, телекоммуникаций, происходящее в настоящее время, приводит к пересмотру разработанных методов и созданию новых алгоритмов на основе принципиально новых подходов.

Рост приложений систем идентификации связан с развитием телекоммуникаций и новыми возможностями общения человека и машины [1,2]. Распознавание спикера - это общий термин, используемый для двух связанных проблем: идентификация спикера и верификация спикера. В задачу идентификации входит узнать неизвестного спикера из набора N известных спикеров. В задаче верификации имя спикера дано распознающей системе и цель принять

или отвергнуть провозглашенную идентичность. Современные достижения в этой области имеются в обзоре [3]. Входом в компьютерную систему идентификации спикера является разделенные на временные окна данные речи, а выходом является индекс идентифицированного спикера. В этой задаче имеется три важных составляющих [4]: выделение признаков реального сигнала, создание модели спикера и алгоритм нахождения соответствия. Целью первой составляющей является нахождение набора векторов признаков входного речевого сигнала специфических для спикера. Модель спикера создается из этих векторов для каждого спикера. Алгоритм соответствия производит сравнение моделей спикеров. Выделение признаков является наиболее критичной компонентой системы и также является намного более сложной частью, чем алгоритм соответствия. Оцифрованный сигнал, соответствующий речи спикера, содержит набор высоких частот, модулированный по амплитуде функциями типа всплесков. Частота всплесков намного меньше несущей частоты сигнала. Эти всплески соответствуют характерным гортанным особенностям данного человека и представляют его уникальную характеристику. Предполагается, что эта характеристика во многом не зависит от текста, произносимого спикером и, в отличие от проблемы автоматического распознавания речи, необходимо выделить текстонезависимые признаки, характеризующие вокальный тракт. Как правило, под признаками имеется в виду либо коэффициенты временного фильтра, выделяющего всплеск, или коэффициенты детализации вейвлета или спектральные характеристики всплесков. Выбор способа зависит от приложений. В любом случае, необходимо параметрическое описание сигнала, характеризующего его акустические свойства. Здесь мы будем рассматривать нахождение определенного вектора признаков из речевого сигнала путем спектрального анализа на коротком промежутке времени. Для каждого короткого промежутка времени, фрейма речи, (обычно 20-30 мс) процесс выделения признаков дает вектор, который характеризует акустические характеристики этого фрейма. Далее в множестве векторов признаков выделяют кластеры.

Способы отбора признаков варьируется в зависимости от приложений. Однако, в общем случае, необходимо, чтобы непохожие акустические вектора могли бы быть отделены друг от друга в пространстве признаков, а соответственно подобные вектора были бы близки друг к другу. В терминах распознавания образов, вариация векторов внутри класса должна быть меньше для обеспечения высокой точности распознавания. Поэтому необходимо, чтобы вектора формировали сепарабельные кластеры в пространстве признаков. Центроиды этих кластеров представляют собой вектора–эталоны

данного спикера. Роль квантования векторов рассматривается в работе [4]. Авторы сравнили производительность различных алгоритмов кластеризации и выявили влияние сложности модели. По близости предъявляемого неизвестного сигнала к вектору-эталону происходит идентификация сигнала. Качество микрофонной записи предъявляемого сигнала и наличие помех во временной записи ухудшает точность идентификации.

В настоящей работе проводится исследование идентификации методом кластеризации (нахождения центроидов) в пространстве кепстральных коэффициентов. Сначала рассматриваются "идеальные" условия регистрации речи спикеров: одинаковые фразы, произнесенные разными спикерами в одинаковых условиях. Для этого случая выясняются наилучшие параметры и признаки в смысле качества идентификации. Наши исследования подтверждают тот факт, что для идеальных условий эксперимента можно добиться стопроцентной идентификации. Однако полученные данные близости векторов признаков говорят о том, что расстояния внутри класса немногим меньше расстояний между классами. Огибающие-всплески сигнала довольно близки друг к другу. Важным свойством любой системы идентификации является ее робастность к внешним помехам [5]. Это приводит к необходимости анализа робастности получаемого результата идентификации к условиям проведения экспериментов. Мы проводим эксперименты с двумя типами помех. Первый - это пассивные помехи, связанные с ухудшением записи (расстояние и угол между спикером и микрофоном, специальное ухудшение записи). Второй – активные помехи, когда на этапе микрофонной записи данного спикера присутствует речь других спикеров.

Система идентификации спикера

Мы рассматриваем систему идентификации, в которой имеется две фазы: обучение и распознавание. На стадии обучения, математическая модель строится для каждого спикера из выборок их речей, и эти модели хранятся в базе данных. На стадии распознавания, анализируются речевые данные неизвестного спикера, и ищется наиболее подходящая модель.

На рис. 1 показана типичная временная последовательность $s(t), t = 1,..., N_t$, N_t число временных отсчетов, соответствующая речевому сигналу, записанному с микрофона. Последовательность $s(t)$ соответствует произнесению фразы, содержащей 171 слово (порядка 1000 букв). Частота записи (сэмлирование) 8 KHz, полная последовательность содержит 547620 точек по времени. Хотя такие

временные последовательности $s(t_i)$ заметно отличаются у разных спикеров, сравнивать их непосредственно в некоторой норме не имеет смысла по следующим причинам. Запись $s(t_i)$ отражает смысл произнесенной фразы (произносимый текст) и индивидуальные особенности речевого тракта спикера. Более того, один и тот же спикер может произнести ту или иную фразу по разному. Все эти эффекты отражаются на форме $s(t_i)$. Кроме того, непосредственное сравнение временных последовательностей такой длины накладывает сильные ограничения на время работы компьютерной системы идентификации. Поэтому задача состоит в том, как на основе статистики выделить немногие характерные акустические особенности речи данного спикера, т.е. сократить информацию и оставить только те признаки, которые позволяют отличать одного спикера от другого. Эта статистика набирается не из разных полных (длительных) последовательностей, а из коротких частей этих последовательностей – временных окон (фреймов). Длительность фреймов, как правило, настолько мала, что соответствует при звучании произнесению одной буквы. Сигнал делится на короткие фреймы фиксированной длины, обычно 20-30 мс. После разделения на фреймы эти короткие "подсигналы" рассматриваются как независимые сигналы. Для каждого фрейма вычисляется вектор признаков конечной длины, который описывает акустическое поведение фрейма. На рис 2 (а, б) показаны сигналы, составляющие два временных фрейма по 30 мс, и содержат 480 временных точек. Рис. 2а и рис. 2б соответствуют произнесению первым и вторым спикером. Считается, что этот временной промежуток достаточен для того, чтобы выделить признаки сигнала, характеризующие данного спикера. Хорошо виден набор повторяющихся колебаний, однако не они, а огибающая этих колебаний является важной характеристикой для распознавания. В качестве признаков может использоваться, например, конечный набор коэффициентов Фурье сигнала на фреймах или набор кепстральных коэффициентов (как правило, 12-ть коэффициентов). В любом случае анализ речевого сигнала основывается на спектральном анализе на коротких фреймах. На рис. 3 (а, б) для тех же спикеров представлены характерные спектрограммы сигналов на фреймах, показанных на рис. 2 (а, б). Заметим, что при произнесении одним спикером разных текстов спектрограмма существенно меняется, но частотная модуляция в среднем остается одинаковой. На фреймах формируются вектора признаков.

Опишем процедуру вычисления векторов признаков из данного речевого сигнала $s(t)$. Наиболее широко используемые признаки в

системах идентификации спикера выводятся из кепструма [3,4,6]. Фуруи [3] первым применил кепстральный анализ в распознавании спикера.

Вначале речевой сигнал обрабатывается с помощью фильтра, выделяющего высокие частоты. Это делается из-за хорошо известного факта, что высокие частоты содержат больше спикер-зависимой информации, чем низкие частоты. Мы используем фильтр с передаточной функцией равной:

$$H(z) = 1 - \alpha z^{-1} \qquad (1),$$

где z комплексная переменная.

Затем, как указывалось выше, исходный сигнал разбивается на фреймы $s_j(\tau)$, где $j = 1,...,L$, L число фреймов, $\tau = 1,...,N_\tau, N_\tau$ число отсчетов в одном фрейме. Затем каждый фрейм умножается на оконную функцию Хэмминга. Тем самым ликвидируются разрывы в конечных точках, что нежелательно для частотного анализа.

Создание речи может быть хорошо аппроксимировано моделью источник-фильтр введенной Фантом [7]. Согласно этой модели, речевая волновая форма есть результат двух независимых компонент: сигнала источника, производимого бронхами и фильтром вокального тракта, который выделяет определенные частоты сигнала источника. Обозначим последовательность возбуждения источника через $e(t)$, а сигнал фильтра вокального тракта через $v(t)$. Результирующая речевая волновая форма, есть свертка этих двух сигналов:

$$u(t) = \sum_{l=1}^{L} v(l) e(t-l) \qquad (2)$$

В частотной области свертка переходит в произведение:

$$U(\omega) = E(\omega) \cdot V(\omega) \qquad (3)$$

Кепструм сигнала $u(t)$, $t = 1,...,N_t$ вычисляется по следующей формуле:

$$R_u(n) = \mathrm{Re}\left[\frac{1}{N_t} \sum_{l=1}^{N_t} e^{i\frac{2\pi(l-1)(n-1)}{N_t}} \ln\left\{ \left| \sum_{t'=1}^{N_t} u(t') e^{-i\frac{2\pi(l-1)(t'-1)}{N_t}} \right| \right\} \right] \qquad (4)$$

Фундаментальная идея в вычислении кепструма в распознавании спикера состоит в подавлении характеристик сигнала, создаваемого бронхами человека, потому что они содержат намного меньше информации об идентичности спикера, чем характеристики вокального тракта. На практике, точное выделение этих двух

нелинейно-смешанных сигналов $e(t)$ и $v(t)$ невозможно, но кепструм дает хорошую аппроксимацию параметров сигнала, которые характеризует вокальный тракт. Детали вычисления кепструма можно найти в [8]. Результатом деконволюции является последовательность кепстральных коэффициентов, где *M* есть количество желаемых коэффициентов. Коэффициент $R_u(0)$ соответствует полной энергии фрейма, и поэтому не содержит информацию о спикере. Обычно, $R_u(0)$ отбрасывается или используется для нормализации. На рис.4 представлены кепстральные коэффициенты речевого сигнала в одном фрейме для сигналов, показанных на рис. 2 (а, б): сплошная кривая соответствует первому спикеру, пунктирная соответствует второму спикеру.

Таким образом, последним шагом процедуры вычисления векторов признаков заданного сигнала $s(t)$ является вычисление кепстральных коэффициентов на фреймах этого сигнала $R_{s_j}(n)$. Вектор признаков будем обозначать $x_j = \{R_{s_j}(n)\}, n = 1,...,M, j = 1,...L$. L число фреймов (мощность статистики).

Выделенное множество векторов признаков обрабатывается далее с использованием процесса векторного квантования для разыскания кластеров в пространстве признаков и для сокращения количества данных. Входными данными процесса векторного квантования является множество векторов признаков X и выходом является модель спикера $C = \{c_k\}, k = 1,..., K$, которая состоит из центроидов кластеров, выделенных в множестве X. K-число кластеров, выделенных из множества векторов признаков *X*.

Пусть число возможных спикеров равно N. Фаза обучения состоит в построении моделей для всех спикеров, участвующих в идентификации.

Процедура идентификации спикера по заданной записи $s(t)$ формулируется следующим образом:

найти набор векторов признаков $X = \{x_j\}, j = 1,..., L$, соответствующих записи $s(t)$

для модели каждого *i*-го спикера C_i вычислить отклонение $D_i = d(X, C_i)$ между *X* и C_i.

Определить индекс неизвестного спикера I_d, как характеристику наименьшего отклонения, т.е.

$$I_d = \arg \min_{i'=1,\ldots,N} \{D_{i'}\} \qquad (5)$$

Мера отклонения *d* на втором шаге аппроксимирует несхожесть между моделью *i*-го спикера C_i и множеством *X* векторов признаков неизвестного спикера, вычисленных по заданной записи $s(t)$. Мера отклонения определяется следующем функционалом:

$$d(X, C_i) = \frac{1}{L} \sum_{j'=1}^{L} \min_{k'=1,\ldots,K} d_E\left(x_{j'}, c_i^{k'}\right) \qquad (6)$$

где d_E – расстояние в усредненной Евклидовой метрике:

$$d_E(x, y) = \sqrt{\frac{1}{N} \sum_{l=1}^{N} (x_l - y_l)^2} \qquad (7)$$

Заметим, что в стадии обучения мы создаем модели для спикеров, а в стадии распознавания мы выполняем прямое сравнение между набором векторов признаков и моделями известных спикеров.

Когда мы проводим кластеризацию векторов признаков, мы находим эффективные средние значения разных акустических признаков на коротком интервале времени. Модели различных спикеров могут содержать некоторые вектора признаки очень близкие друг к другу, в то же время в моделях будут существовать уникальные вектора признаки, которые позволят разделить модели различных спикеров.

Возникают два важных вопроса в векторном квантовании: выбор метода генерации модели спикера и размер модели.

Проблема кластеризации определяется следующим образом. Для данного набора векторов признаков $X = \{x_1, \ldots, x_L\}$ требуется разделить набор данных на $K<<L$ кластеров таким образом, чтобы близкие вектора в смысле Евклидова расстояния были сгруппированы в одном кластере. Модель *i*-го спикера $C_i = \{c_i^1, c_i^2, \ldots, c_i^K\}$ далее может быть сформирована из векторов, которые являются векторными средними каждого кластера. Мы использовали один из наиболее часто используемых методов кластеризации метод К-средних. В работе [4] указано, что 100 % точность распознавания достигается при разбиении на 64 кластера и больше.

Результаты идентификации в «идеальном» эксперименте

Было исследовано качество работы алгоритма в "идеальных условиях". Под "идеальными условиями" понимается отсутствие шума и различных помех при создании записей спикеров. Для проведения экспериментов была создана база данных, состоящая из записей фраз спикеров, разделенных на две группы: к первой группе относятся фразы, на основе которых проводится вычисление моделей спикеров, ко второй группе относятся фразы, используемые для идентификации спикера.

Опишем создание речевой базы данных. Четыре спикера (3 мужчины и одна женщина) в одинаковых условиях произнесли три фразы: Текст 1, Текст 2, Текст 3. При произнесении фразы спикер находился напротив микрофона на расстоянии 30-40 см. Записи делались при частоте сэмплирования 44,1 kHz. Для уменьшения объема обрабатываемой информации частота сэмплирования понижалась до 8kHz. Средняя продолжительность фраз составляла 77.25 с, 45.5 с, 52.75 с соответственно. Фразы содержали соответственно 1035, 666, 779 букв. Таким образом, в получении статистики при обучении участвовало в среднем 3300 фреймов при обучении на первой фразе, 2200-2400 фреймов при обучении на второй и третьей фразах.

Для оценки точности качества работы алгоритма исследовались два следующих показателя. Первый показатель оценивает точность аппроксимации голоса *i*-го спикера созданной моделью C_i. Этот параметр есть оценка близости поданной записи $s(t)$, характеризуемой множеством векторов признаков $X_{s(t)}$, к модели *i*-го спикера C_i:

$$\varepsilon^i_{abs}(X) = d(X_{s(t)}, C_i) \qquad (8),$$

функционал $d(\cdot,\cdot)$ определяется формулой (6). Второй показатель оценивает робастность идентификации *i*-го спикера:

$$\Omega^i_{abs}(X) = \min_{i' \neq i}\{d(X_{s(t)}, C_{i'})\} - d(X_{s(t)}, C_i) \qquad (9)$$

Если значение этого параметра положительное, то *i*-ый спикер идентифицируется в записи $s(t)$. Чем больше значение этого параметра, тем более надежна идентификация *k*-го спикера.

Вероятность верной идентификации *i*-го спикера, вычисляется по формуле:

$$P\left(\Omega^{i}_{abs} > 0\right) = \int_{0}^{\infty} \frac{1}{\sqrt{2\pi D\left\{\Omega^{i}_{abs}\right\}}} e^{-\left[\frac{\Omega^{i}_{abs} - E\left\{\Omega^{i}_{abs}\right\}}{D\left\{\Omega^{i}_{abs}\right\}}\right]^{2}} \cdot 100\, d\Omega^{i}_{abs} \qquad (10),$$

где $E\left\{\Omega^{i}_{abs}\right\}$ - математическое ожидание, а $D\left\{\Omega^{i}_{abs}\right\}$ - дисперсия величины Ω^{i}_{abs}. Вероятность идентификации в идеальных условиях составила 100 % для всех спикеров.

Одним из основных этапов вычисления модели заданного спикера является процесс кластеризации в пространстве выделенных векторов признаков. Вопрос качества кластеризуемости векторов признаков речи очень важен при разработке систем идентификации спикера. Для проведения кластеризации мы использовали метод К-средних. При использовании этого метода надо задавать число выделяемых кластеров K. Мы исследовали зависимость качества кластеризации от K. Для оценки качества кластеризации используется параметр F_{mecm} [9], который в общем виде есть отношение средней дисперсии элементов внутри кластера к дисперсии между центроидами:

$$F_{mecm} = \frac{\sum_{k'=1}^{K} d_E\left(\frac{1}{K}\sum_{k''=1}^{K} c^{k''}, c^{k'}\right)^2}{\sum_{k'=1}^{K} \frac{1}{n_{k'}} \sum_{i:g_i=k'} d_E\left(x_i, c^{k'}\right)^2} \qquad (11),$$

n_k -число элементов в k-ом кластере.

Максимальные значения параметра соответствуют наилучшему качеству кластеризации. На рис 5 показана зависимость значений этого параметра от числа кластеров, на которые мы разделяем множество векторов признаков для разных спикеров. Значение параметра немонотонно возрастает в зависимости от числа кластеров K. Начиная со значений параметра $K \in [16,64]$, скорость возрастания кривых уменьшается. Выбор количества кластеров должен определяться необходимой точностью идентификации и ограничениями на сложность метода. 100 % точность идентификации достигается при разбиении на 64 кластера.

Значение параметра F_{mecm} много меньше единицы. Это означает, что средняя дисперсия элементов внутри кластеров больше, чем дисперсия центроидов, то есть кластеры расположены очень близко друг к другу по сравнению с их диаметрами. На рисунке 6 (а, б) показана двумерная проекция функции распределения векторов

признаков по первым двум кепстральным коэффициентам. Стрелки 1,2 указывают на характерные кластеры, разнесенные друг от друга. Стрелка 3 указывает на общий для этих спикеров кластер.

Рассмотренный метод показал 100% идентификацию спикера в "идеальных условиях".

Идентификация в условиях пассивных помех

Вначале рассмотрим робастность идентификации спикера при наличии пассивной помехи. Пассивной помехой считается препятствие, расположенное между спикером и микрофоном, которое искажает характеристики голоса спикера. На пример, пассивной помехой может являться ткань, через которую говорит спикер.

Был поставлен следующий эксперимент. Спикер произносит одну и ту же фразу через ткань, свернутую в несколько слоев Z. Обучение осуществляется на фразе, содержащей 142 слова (около 1000 букв), произнесенной без помех. Для идентификации используется фраза, содержащая 71 слово (483 буквы).

Робастность идентификации оценивалась показателем $\Omega_{abs}(X)$. На рисунке 7 показана зависимость $\Omega_{abs}(X)$ от числа слоев ткани Z. Из графика видно, что робастность метода ухудшается с увеличением числа слоев Z, и при числе слоев равном 16 робастность уменьшается в 2 раза. Тем не менее, спикер верно идентифицируется.

Идентификация в условиях активной помехи

Под активной помехой мы понимаем присутствие при записи речи спикера другого сигнала, который искажает характеристики голоса спикера. Как активную помеху мы рассматривали присутствие при записи речи другого спикера. Таким образом, требуется идентифицировать спикера в смеси речей.

Мы исследовали робастность метода идентификации при наличии в записи речи идентифицируемого человека голоса другого человека. Параметром этой зависимости мы используем отношение энергии сигнала идентифицируемого человека (полезного сигнала) к энергии сигнала речи другого человека. Он вычисляется по следующей формуле

$$SNR = \frac{\sum_t s_1^2(t)}{\sum_t s_2^2(t)} \qquad (12),$$

где $s_1(t)$ сигнал идентифицируемого спикера, $s_2(t)$ сигнал шума.

Спикеры располагаются на расстоянии 0.4 м друг от друга и 0.28 м от микрофона. Робастность метода оценивалась показателем $\Omega_{abs}(X)$. На рисунке 8 изображена зависимость параметра $\Omega_{abs}(X)$ от *SNR* (сплошная линия). Значению параметра $\Omega_{abs}(X)$ при идентификации без активной помехи соответствует горизонтальная пунктирная прямая. Из графика следует, что метод работает неустойчиво в присутствии активной помехи. В области, где энергия шума больше энергии полезного сигнала, идентификации отсутствует. При увеличении энергии полезного сигнала метод начинает верно идентифицировать, однако робастность идентификации низкая. Расстояние между кривой $\Omega_{abs}(X)$ в присутствии активной помехи и кривой $\Omega_{abs}(X)$ без помехи слабо меняется, что говорит о наличии неустранимой ошибки идентификации, вносимой активной помехой.

Заключение

В работе рассмотрена проблема потери точности идентификации спикера по записи его речи в присутствии помех – пассивных и активных. Использован широко применяемый метод идентификации, основанный на кластеризации данных в пространстве кепстральных коэффициентов.

В идеальных условиях метод показал 100% точность идентификации. Исследована структура кластеров, образуемых векторами признаками речи. Получено, что в речи спикеров присутствуют общие кластеры, и кластеры, уникальные для конкретного спикера. Это приводит к идеи введения весов в функционал (2) с целью увеличения влияния центроидов уникальных кластеров и уменьшения влияния центроидов общих классов при вычислении близости записей. При исследовании проекций было получено, что кластеры имеют эллиптическую форму. Перспективными выглядят подходы для идентификации, связанные с применением разделяющих поверхностей.

Исследована зависимость робастности метода в условиях пассивной помехи. Показано, что при усилении пассивной помехи, робастность метода ухудшается.

Проведено исследование зависимости робастности метода в условиях активной помехи, на примере идентификации спикера в смеси. Показано, что наличие активной помехи при записи спикера может привести к неустранимой ошибке идентификации и сделать невозможным идентификацию.

Работа выполнена при поддержке гранта РФФИ 02-07-90130, а так же в рамках работы в студенческой лаборатории Intel-MSU, ВМК, МГУ.

Иллюстрации

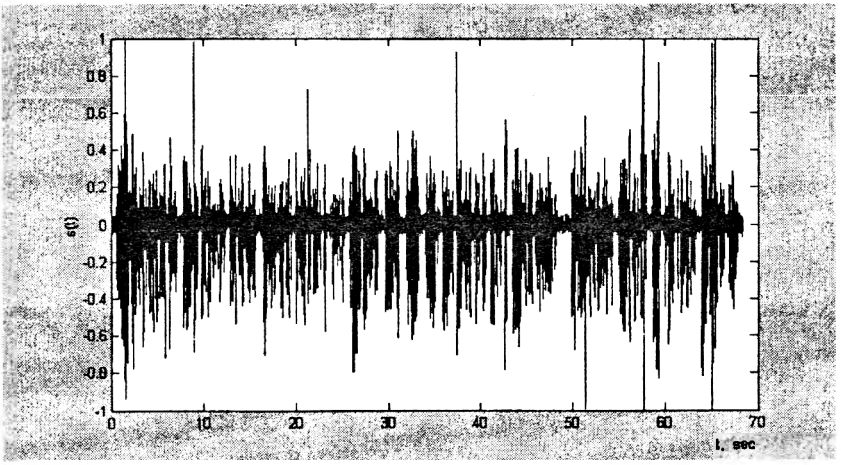
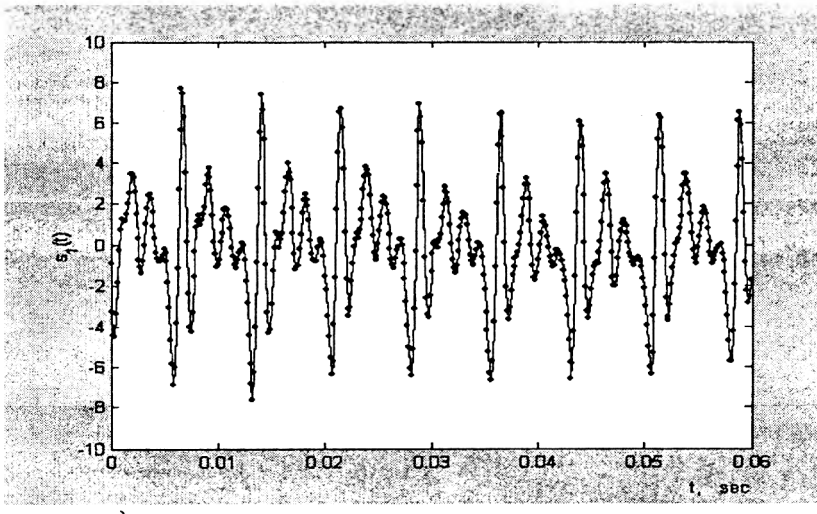
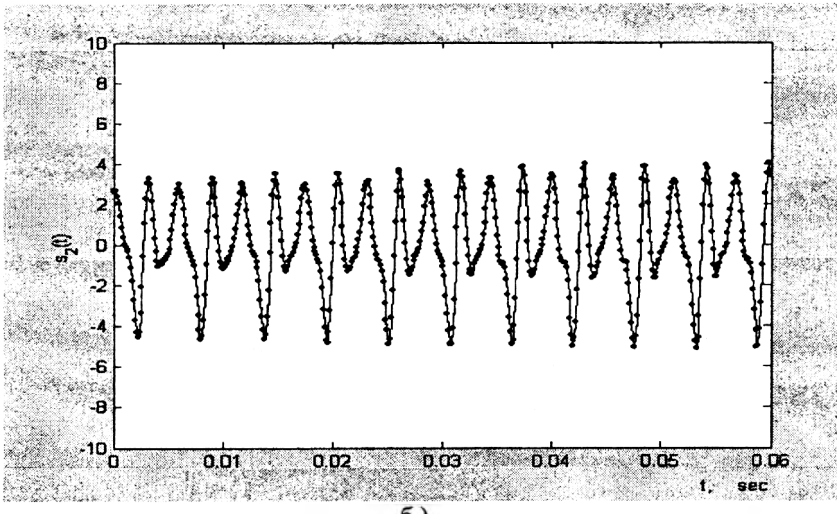


Рис. 1. Полный речевой сигнал спикера. Частота сэмплирования 8000 Hz. Сигнал соответствует тексту, содержащему 171 слово, порядка 1000 букв.

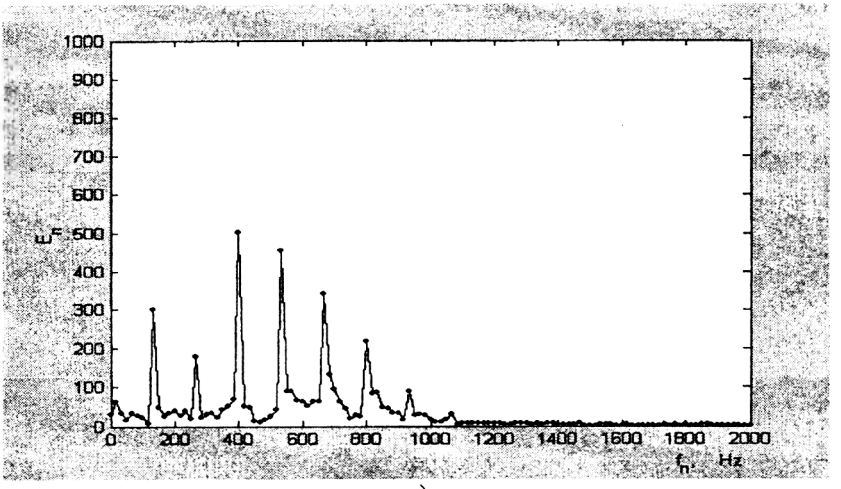


а.)

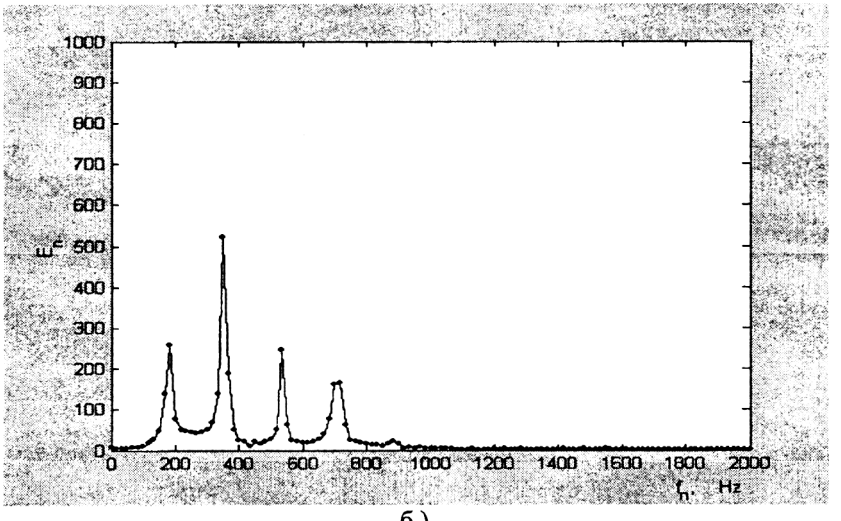


б.)

Рис. 2 (а, б). Временные сигналы спикеров в одном фрейме. (а) $s_1(t)$ - сигнал 1-го спикера; (б) $s_2(t)$ - сигнал 2-го спикера.



а.)



б.)

Рис. 3 (а, б). Спектрограммы речевого сигнала первого (а) и второго (б) спикеров в одном фрейме.

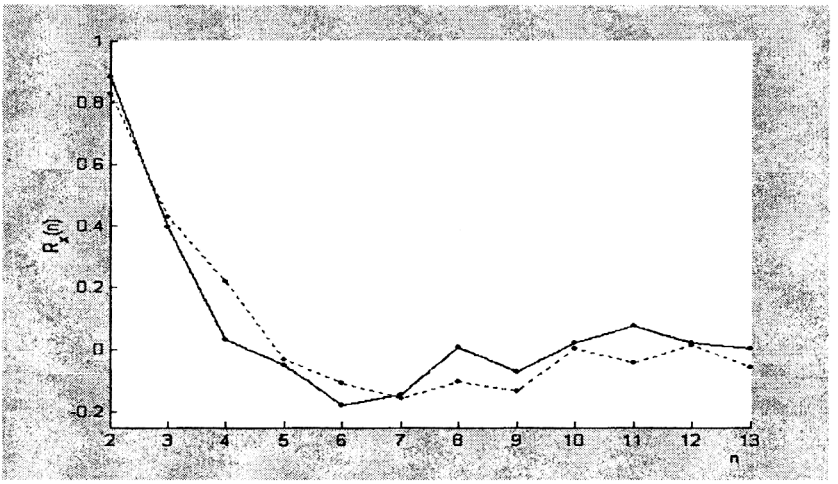


Рис. 4. Кепстральные коэффициенты речевого сигнала в одном фрейме: сплошная кривая соответствует первому спикеру, пунктирная соответствует второму спикеру.

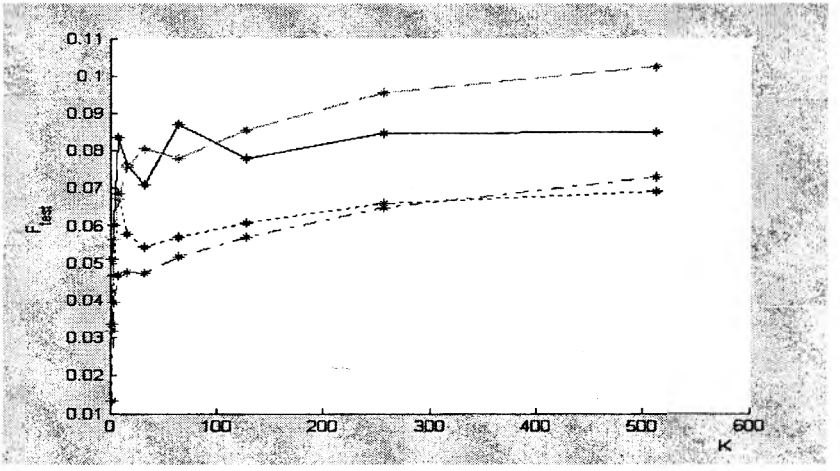
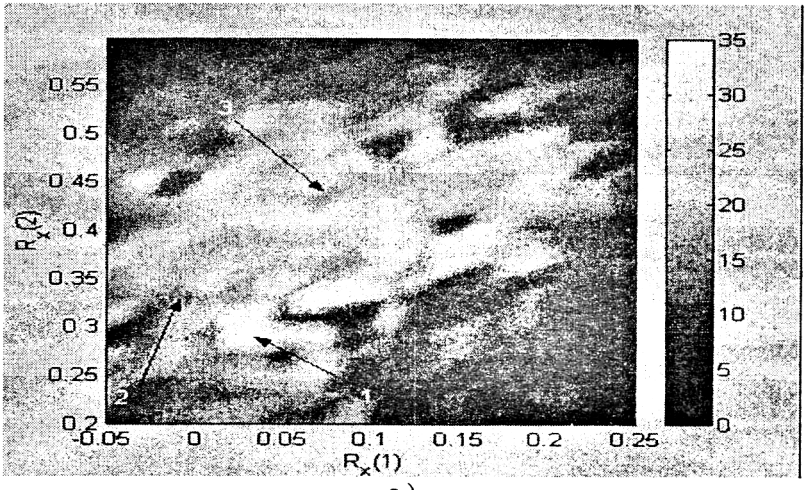
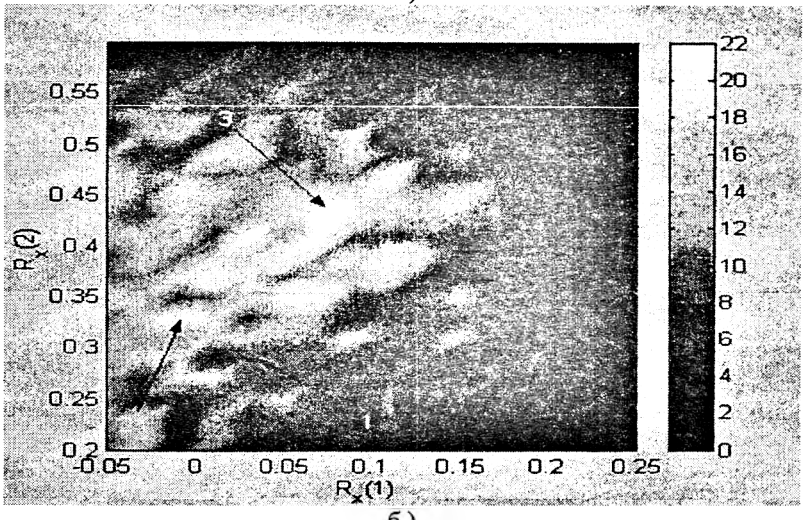


Рис. 5. Зависимость F_{mecm} от числа кластеров K для разных спикеров.



а.)



б.)

Рис. 6 (а, б) Проекция функции распределения векторов признаков по первым двум кепстральным коэффициентам для разных спикеров: (а) – первый спикер. (б) – второй спикер. Стрелки 1,2 указывают на разные кластеры. Стрелка 3 указывает на общий кластер.

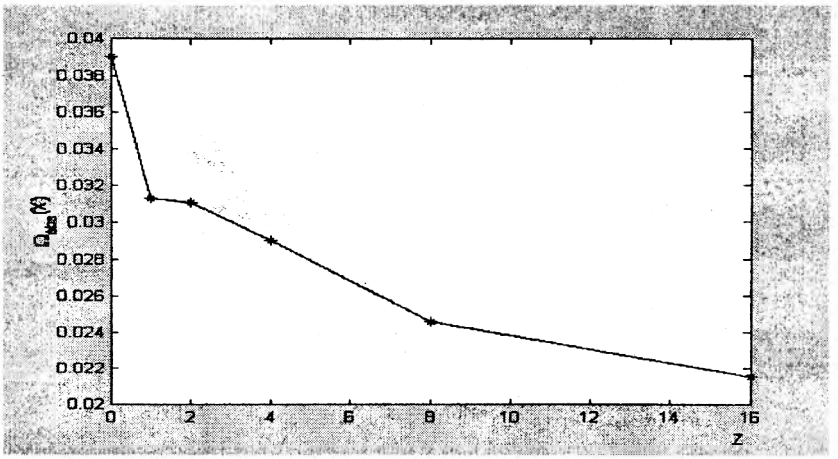


Рис. 7. Потеря точности идентификации в условиях пассивных помех в зависимости от числа слоев Z.

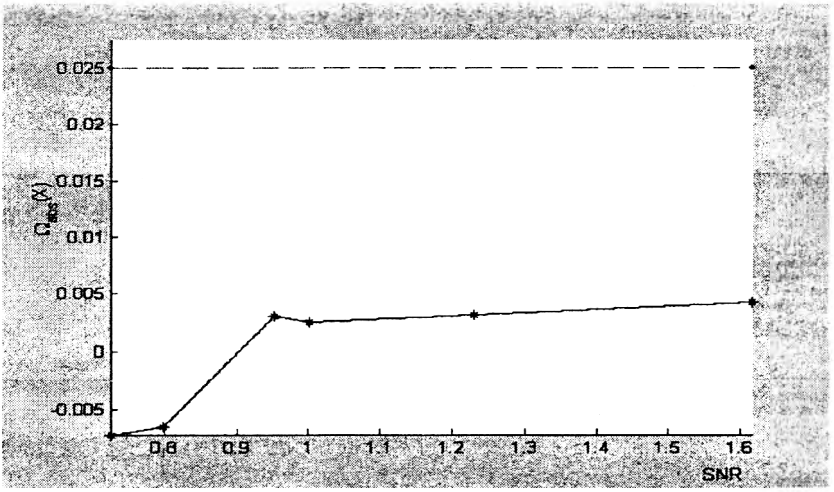


Рис. 8. Потеря точности идентификации в условиях активной помехи. Зависимость робастности метода от SNR.

Литература

1. Deller, J. R., J. H. L. Hansen and J.G. Proakis, *Discrete-Time Processing of Speech Signals*. IEEE Press, 2000.

2. M. Brandstein and S. Griebel , "Nonlinear, Model-Based Microphone Array Speech Enhancement," In J. Benesty and S. Gay, editors, *Theory and Applications of Acoustic Signal Processing For Telecommunication*, Kluwer Academic Publishers, 2000.

3. Furui, S.: Comparison of Speaker Recognition Methods Using Statistical Features and Dynamic Features, IEEE Transactions on Acoustics, Speech, and Signal Processing, No.3, June 1981

4. Kinnunen T., Kilpelainen T., Franti P. Comparison of clustering algorithms in speaker identification., *IASTED Int. Conf. on Signal Processing and Communications (SPC'00)*, Marbella, Spain, 222-227, 2000.

5. D.L. Wang, G.J. Brown. Separation of Speech from Interfering Sounds Based on Oscillatory Correlation. IEEE Transactions on neural networks, vol. 10, No. 3, May 1999

6. T. S. Chang, S.D. Van Hooser. Two new methods for speaker recognition using cepstral analysis., CNS-Spring, 1996.

7. Fant G.: Acoustic Theory of Speech production. (Moution: The Hague, 1960).

8. Kinnunen T., Franti P.: Is speech data clustered? – statistical analysis of cepstral features., *European Conf. on Speech Communiation and Technology, (EUROSPEECH'2001)*, Aalborg, Denmark, Vol. 4, pp. 2627-2630, September, 2001.

9. Özgür Devrim Orman. Frequency analysis of speaker identification performance. Master of Science Thesis, Bogazici University, 2000.

Раздел IV
Распараллеливание вычислений

Сальников А. Н., Сазонов А. Н., Карев М. В.

Прототип системы разработки параллельных программ для гетерогенных многопроцессорных систем

Введение

Программы, написанные для машин последовательной архитектуры напрямую невозможно использовать на машинах с параллельной архитектурой, не сводя эффективность исполняемого кода программы практически в ноль. Для каждой параллельной архитектуры существует своя специфика написания эффективных параллельных программ, так что программы, эффективно исполняющиеся на одной параллельной вычислительной системе, в подавляющем большинстве случаев непригодны, а если и пригодны, то неэффективны на другой. Таким образом, решение сложных задач на вычислительной системе с параллельной архитектурой связано с необходимостью предварительного анализа алгоритма и выбора оптимального способа организации параллельных вычислений.

В текущий момент времени существует достаточно большое число компиляторов с различных языков параллельного программирования. В качестве примера можно привести такие языки, как ***mpC***, ***HPF***, ***Sc++***, ***OpenMP***, ***DVM***. Все они предназначены для написания параллельных приложений. Данные языки, а так же подобные им требуют некоторых сведений от пользователя относительно способа распараллеливания программ, что в некоторых случаях может быть затруднительно.

Решение задачи автоматизации параллельного программирования

По последовательному коду программы написанной на языке программирования *C* строится граф алгоритма, где вершины символизируют действия, а дуги зависимость по данным. Производится тестирование многопроцессорной системы на скорость передачи сообщений между отдельными процессорами многопроцессорной

системы, а так же тестирование производительности процессоров. Граф преобразуется в параллельную программу, например программу, написанную на языке *C++* с *MPI* вызовами. Строится расписание выполнения полученной параллельной программы на многопроцессорной системе. Параллельная программа компилируется и может быть выполнена на многопроцессорной системе.

На рис. 1 представлены компоненты системы, а также связи между ними. Стрелочка означает передачу файла на вход компоненте системы.

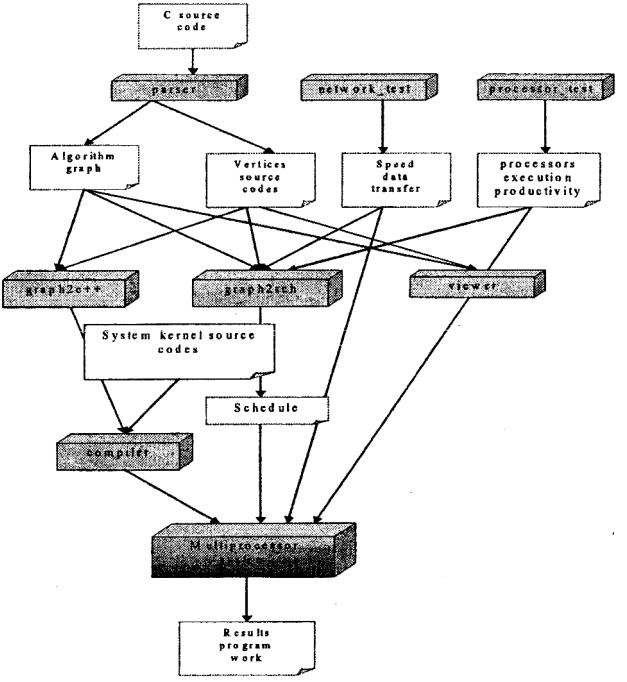


Рисунок. 1 Компоненты системы

parser – по С программе строит граф алгоритма.

graph2c++ – по графу алгоритма строит программу на C++ с MPI вызовами.

graph2sch – по графу алгоритма и результатам тестов строит статическое расписание выполнения программы на многопроцессорной системе.

network_test – производит тестирование коммуникаций в многопроцессорной системе.

processor_test – производит тестирование процессоров в многопроцессорной системе.

viewer – отображает граф алгоритма на мониторе компьютера.

compiler – обычный компилятор MPI с помощью которого компилируется параллельная программа.

Понятие взвешенного графа алгоритма

Под *взвешенным графом алгоритма* понимается граф удовлетворяющий следующим требованиям:

1) Ориентированный граф, с пометками на дугах и вершинах без ориентированных циклов. Направление дуги задает порядок следования операторов, метка - передаваемые данные.

2) Отсутствуют дуги между вершинами одного яруса.

3) Отсутствуют дуги "снизу-вверх" - от вершин в ярусе с большим номером к вершинам яруса с меньшим номером. Ярусы нумеруются по возрастанию. Метки (операторы) вершин, расположенных в одном ярусе могут быть выполнены одновременно. Ярусы нумеруются естественным образом от вершин источников, вершин в которые не входит ни одной дуги, к вершинам стокам, вершин из которых не исходит ни одной дуги.

Вершины графа — вычисления, а дуги — информационные зависимости между ними по данным. При этом дуга направлена от вершины источника данных, к вершине приемнику данных. Зависимости по данным фиксируют необходимый порядок выполнения операций. Существует три типа таких зависимостей:

Прямая зависимость «out-in»

Обратная зависимость «in-out»

Зависимость по выходам «out-out»

Антизависимость по данным

Такой тип зависимости характеризуется тем, что оператор должен прочитать старое значение переменной, до того как оно изменится, другими словами чтение и запись нельзя менять местами, и вообще изменять их взаимный порядок. (рис. 2)

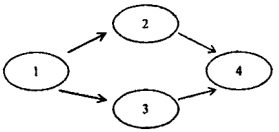


Рисунок. 2 антизависимость

Множество дуг антизависимостей:

$$E_2 = \{ (l,k) \mid k > l,\ Out(k) \cap In(l) \neq \varnothing \}$$

Стоит отметить, что такой тип зависимости не вызывает передачи данных, и дуги обратной зависимости всегда имеют нулевой вес.

Зависимость по выходным данным

Последний тип завимости – зависимость по выходам, неочевиден и требует подробного рассмотрения.

1. b=0;
2. b=1;
3. c=b+2;
4. a=b+1;

В этом случае для корректной работы программы узел **2** должен выполняться после **1**-го узла графа алгоритма.

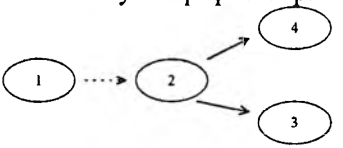


Рисунок 3. Зависимость по выходам 1

Узлы **3** и **4** в любом случае могут быть исполнены независимо, но строго после узла **2**. (рис. 3) Если же предположить такую последовательность выполнения узлов, как на рис. 4,

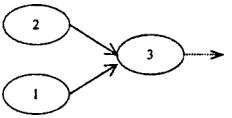


Рисунок. 4. Зависимость по выходам 2

то неизвестно какое значение b будет считано в 3-м узле. Множество дуг, отражающих зависимость по выходам:

$$E_3 = \{ (l,k) \mid k > l,\ Out(k) \cap Out(l) \neq \varnothing \}$$

Итак, вершина V_1 выполняется после V_2, если дуга (V_1,V_2) принадлежит транзитивному замыканию бинарного отношения $E = E_1 \cup E_2 \cup E_3$

Для полного анализа программы и оптимального распараллеливания алгоритма на многопроцессорной системе, необходима дополнительная информация. Вершинам графа алгоритма приписывается *вес*, равный объему вычислений производимых в узле (измеряемому, например, в количестве тактов работы процессора). Дуги

снабжаются *весом*, равным размеру передаваемых данных (например, в байтах).

Очевидно, что дуги прямой зависимости всегда имеют ненулевой вес, дуги обратной зависимости всегда имеют нулевой вес, а дуги зависимости по выходам могут иметь как нулевой вес, так и ненулевой, исходя из ситуации. Именно поэтому, этот тип зависимости не является транзитивным следствием из двух других типов.

Параллельная программа в форме графа алгоритма

Граф алгоритма преобразуется в параллельную программу. Операторы, приписанные вершинам графа, преобразуются в функции и могут быть выполнены на произвольном процессоре многопроцессорной системы. Дуги становятся вызовами функций передачи сообщений.

Для эффективного выполнения параллельной программы на многопроцессорной системе граф алгоритма должен обладать следующими дополнительными свойствами:

1. Узел графа алгоритма сложный, в вычислительном смысле, набор операторов. Задача выбора правильного размера узла графа алгоритма, чрезвычайно важна. Слишком лёгкий узел графа алгоритма приведёт к тому, что накладные расходы при вызове его на конкретном процессоре, перекроют по времени тот полезный код, который выполняется в данном узле графа.

2. Операторы узла графа алгоритма явно обращаются только к локальной памяти процессора многопроцессорной системы. Данное свойство является следствием того, что рассматриваются многопроцессорные вычислительные системы с распределённой памятью.

3. Операторы узла графа алгоритма выполняются только в тот момент времени, когда получены все данные, входящие в вершину графа по дугам графа. Данные по дугам графа алгоритма могут приходить в произвольном порядке, асинхронно.

Для корректности построения графа алгоритма по последовательной программе должны выполняться следующие требования:

1. Происходит только детерминированная передача элементов массивов или частей массивов, являющихся непрерывными частями массива, например участок от 10 до 20 элемента массива, а не от 10 до j го, где j имя переменной.

2. Необходимые данные передаются последним менявшим их узлом.

Так как граф алгоритма строится по последовательной программе на языке *C* и по нему автоматически строится код на языке *C++* с *MPI* вызовами, необходимы некоторые соглашения о переменных программы:

1. Все переменные, общие для различных узлов графа, должны быть описаны глобально.
2. В специальном узле графа (root) должна быть описана та часть программы, которая должна быть выполнена на всех процессорах до вызова узлов графа алгоритма.
3. В специальном узле графа алгоритма (tail) должна быть описана та часть программы, которая должна быть выполнена перед завершением программы.
4. Переменные необходимые только данному узлу графа должны быть описаны в поле (head) узла графа алгоритма.

На (рис. 5) представлена схема узла графа алгоритма.

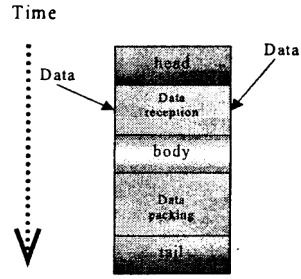


Рисунок. 5 Узел графа алгоритма

head – код содержит описания переменных и начальную инициализацию данных необходимых узлу.

body – код выполняемый после получения всех необходимых данных.

tail – код содержит действия по подчистке памяти и утилизации данных.

Блок, предшествующий блоку ***tail*** и блок, предшествующий блоку ***body*** человеку не видны, код для этих блоков подставляется автоматически программой преобразования графа в параллельную программу на *C++* с включениями *MPI* вызовов.

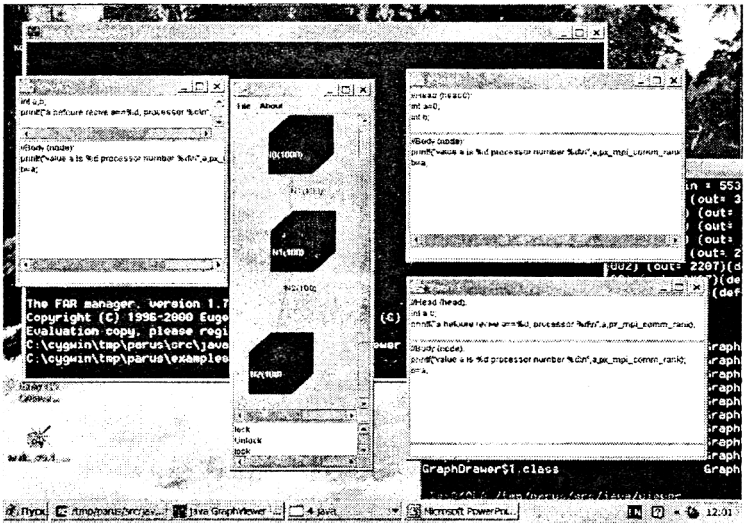


Рисунок. 6 Простой пример программы

Построение расписания

Формулировка проблемы

По полученной на вход программе в форме графа алгоритма и графу многопроцессорной вычислительной системы строится статическое расписание исполнения программы на многопроцессорной системе. При этом расписание строится близкое к оптимальному по времени выполнения программы в форме графа алгоритма на данной вычислительной системе.

Модель расписания

Расписание выполнения задач представляется в виде списка S={si=(Ti, Pi, ti)}, упорядоченного по ti, где Pi - это процессор, исполняющий задачу Ti, начиная с момента времени ti после старта распределенной программы (пример на рис. 7). Стрелки на рисунке означают передачу данных.

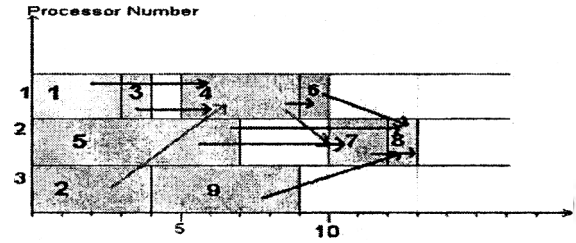


Рисунок 7. Расписание

Задача Ti может выполняться, только если все ее предшественники по направленному графу алгоритма завершили свое выполнение, и все данные, необходимые для ее выполнения, получены процессором Pi.

Интерфейс построителя расписания

На входе требуется: количество узлов графа алгоритма (задач), количество процессоров (узлов графа вычислительной системы), время передачи v единиц данных с процессора Pi на процессор Pj (функция send(Pi,Pj,v)), время работы задачи Ti на процессоре Pj (функция work(Ti,Pj)).

На выходе получаем упорядоченный по порядку выполнения список задач для каждого процессора.

Решение проблемы составления расписания

Задача решается при помощи генетического алгоритма, схема кодирования которого не допускает задания невыполнимых расписаний, например, таких, у которых нарушается условие частичной упорядоченности узлов графа алгоритма. Для подсчета функции качества по генетическому коду особи строится расписание, длина которого, предположительное время работы программы, становится значением функции качества. Таким образом, цель программы построения расписания – минимизировать функцию качества. Желательно найти глобальный минимум, однако это не обязательно, а в случае сложных входных данных и маловероятно. От требуемой точности решения сильно зависит время работы построителя расписания, чем выше точность, тем дольше работает построитель.

Процесс исполнения полученной параллельной программы

Для исполнения полученной параллельной программы в многопроцессорной системе выделяется один специальный управляющий процессор, при помощи которого осуществляется управление деятельностью других процессоров.

В функции управляющего процессора входит:

1. Определение того, какие узлы графа на каком процессоре вызвать.
2. Определение того, по каким рёбрам графа пора вести передачу данных, а по каким нет.
3. Определение того, закончена ли передача данных по тому или иному ребру и посылка сигнала об освобождении буферов памяти.
4. Учёт производительности процессоров и сети.
5. В определённые моменты времени проверка статического расписания полученного до процесса выполнения программы на многопроцессорной системе.
6. В функции остальных процессоров входит:
7. Инициализация передачи данных другим процессорам по команде от управляющего процессора.
8. Вызов узла графа алгоритма на процессоре.
9. приём данных по ребрам от других узлов графа на других процессорах необходимых операторам того узла графа алгоритма, который выполняется на текущем процессоре.
10. Выполнение кода операторов узла графа.
11. Накопление в памяти выработанных данных необходимых другим узлам графа алгоритма.

Для примера представленного на рис. 6 приведён протокол обмена сообщениями между процессорами многопроцессорной системы (рис. 8.)

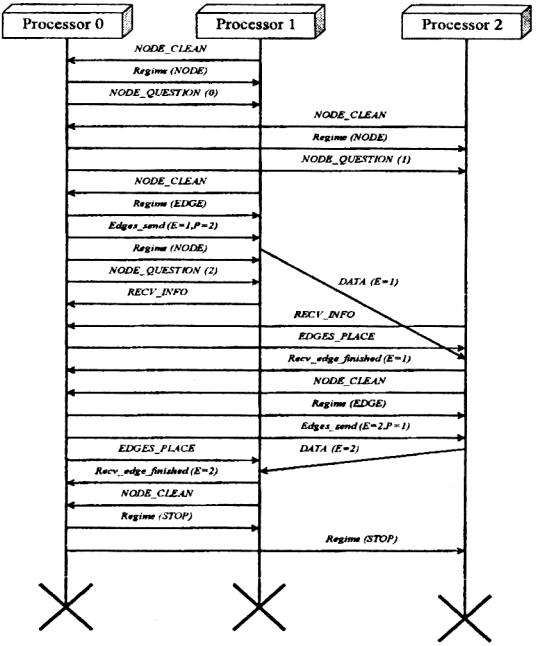


Рисунок 8. Протокол обмена сообщениями между процессорами

Перед тем, как строить расписание выполнения параллельной программы и выполнением самой параллельной программы необходимо получить сведения о производительности многопроцессорной системы. Тесты запускаются в режиме, монопольного захвата процессора.

Производительность процессора проверяется на неоднократном, локальном перемножении матриц фиксированной размерности.

Производительность сети меряется на асинхронном неблокированном многократном обмене данными между всеми процессорами, с постепенно растущей длиной сообщения.

Производительность процессоров представляется в виде вектора, где элемент среднестатистическое время выполнения набора эталонных операций. Производительность сети представляется в виде набора матриц по длинам сообщений, где элемент матрицы – время передачи сообщения между процессорами.

Назначение узла графа алгоритма на процессор

Узлы графа алгоритма располагаются по уровням, и по мере загрузки уровня на процессор следующий уровень добавляется в список узлов графа заявленных на выполнение. Среди всех узлов заявленных на выполнение производится поиск узлов с минимальным временем выполнения на процессоре, где время считается по формуле приведённой ниже. Если минимум достигается для нескольких узлов графа одновременно, то выбирается тот, который заявлен в расписании на выполнение на данном процессоре.

Время выполнения операторов узла графа алгоритма на процессорах многопроцессорной системы вычисляется по формуле.

$$time = \frac{node\_weight \times processor\_weight}{num\_operation} +$$

$$+ \max_{1 \le i \le N} (link\_weight\,(edge\_weight_i)),$$

node_weight – объём выполняемого кода узлом графа алгоритма выраженный в числе эталонных операций.

processor_weight – время за которое процессор выполнит ***num_operation*** эталонных операций.

link_weight(message_length) – функция, вычисляет время за которое будет передано через сеть ***message_length*** байт.

edge_weight – число байт, которое необходимо передать по данному ребру графа алгоритма. Данное число эквивалентно ***message_length***.

edge_weight – подставляется вместо ***message_length*** в функцию ***link_weight***.

N – число входящих в узел графа алгоритма рёбер.

Заключение.

В настоящее время проверена работоспособность системы на тестовых примерах. Разработан набор документации, описывающий работу компонент системы, формат входных данных для компонентов системы, в том числе формат представления графа алгоритма в виде текстового файла, что даёт возможность создавать и редактировать граф алгоритма вручную.

Авторы приносят благодарности члену корреспонденту РАН, профессору Льву Николаевичу Королёву и доктору физ. мат. наук, доценту Нине Николаевне Поповой за помощь в написании статьи и критические замечания в адрес авторов.

Данная работа выполняется в рамках студенческой лаборатории Intel МГУ им. М. В. Ломоносова. При поддержке гранта РФФИ 02-07-90130 и гранта РФФИ 02-07-06104.

Литература.

1. Воеводин В.В. "Информационная структура алгоритмов". - М.: Изд-во МГУ, 1997.- 139с.

2. Костенко В.А. "Способы представления и преобразования расписаний в итерационных алгоритмах", Программные системы и инструменты: тематический сборник факультета ВМиК МГУ (под ред. Л. Н. Королева) вып.2, М., Издательский отдел факультета ВМиК МГУ, 2001, стр. 53-69.

3. Н.А.Коновалов, В.А.Крюков, Ю.Л.Сазанов. C-DVM - язык разработки мобильных параллельных программ. Программирование N 1, 1999.

4. Сальников А.Н. "Разработка инструментальной системы для динамической балансировки загрузки процессоров и каналов связи" Материалы Международного научно-практического семинара "Высокопроизводительные параллельные вычисления на кластерных системах" Нижний Новгород 20-24 ноября 2001 г. стр. 159-167

5. Сазонов А.Н. "Статическое построение расписания выполнения параллельной программы с использованием генетических алгоритмов" Материалы Международного научно-практического семинара "Высокопроизводительные параллельные вычисления на кластерных системах" Нижний Новгород 20-24 ноября 2001 г. стр. 149-159

6. Сазонов А.Н. Сальников А.Н. "Прототип системы разработки приложений обработки экспериментальных данных на гетерогенных многопроцессорных системах" Сборник тезисов Международного семинара "Супервычисления и математическое моделирование" Саров 17-21 июня 2002 г. стр. 71-72.

7. Voevodin V.V. Information Structure of Algorithms.// Intl. Symp. on Optimal Algorithms, Bulgarian Ac. of Sci., Sofia, 1989, p.112-114.

8. Ian T. Foster "Designing and Building Parallel Programs : Concepts and Tools for Parallel Software Engineering." Addison-Wesley, 1995 ISBN 0-201-57594-9

9. Rohit Chandra, Ramesh Menon, Leo Dagum, David Kohr, Dror Maydan, Jeff McDonald "Parallel Programming in OpenMP" ISBN 1-55860-671-8

10. K.Karganov, K. Khorenko, and M.Posypkin, "A mpC Parallel Development Environment", "Proceedings of the International Conference on Parallel and Distributed Procesing Techniques and Applications" (PDPTA'2001), CSREA Press, June 25-28, Las Vegas, Nevada, 2001, pp.1464-1470

11. N.A. Konovalov, V.A. Krukov, S.N. Mihailov and A.A. Pogrebtsov. "Fortran DVM - a Language for Portable Parallel Program Development", Proceedings of Software For Multiprocessors & Supercomputers: Theory, Practice, Experience, 124-133, Institute for System Programming, RAS, Moscow (1994).

12. N. A. Konovalov, V. A. Krukov, Yu. L. Sazanov. C-DVM-A Language for the Development of Portable Parallel Programs. Programming and Computer. Software, v 25, 1-1999, p. 46-55.

13. William Gropp, Ewing Lusk, and Anthony Skjellum. "Using MPI" November 1999 ISBN 0-262-57132-3 350 pp.

14. K. Grant, "An introduction to genetic algorithms", C/C++ Users Journal, March 1995, pp. 45-58

15. David B. Fogel, "Using Evolutionary Programming to Schedule Tasks on a Suite of Heterogeneous Computers", Computers & Operations research, Vol. 23:6, pp. 527-534.

Сальников А. Н.

Некоторые технические аспекты инструментальной системы для динамической балансировки загрузки процессоров и каналов связи

Введение

В современном мире существует множество задач, решение которых требует большого количества ресурсов. В качестве примера можно привести задачу моделирования климата на планете, или задачу построения больших молекул, задачу построения реалистических изображений и компьютерных фильмов. Список таких задач можно продолжать, по-видимому, почти бесконечно. Практика показывает, что использование однопроцессорной системы даже очень высокой производительности для таких задач может быть весьма затруднительным. Однако большинство из этих "тяжёлых" задач можно свести к набору более мелких подзадач, зависимости по данным между которыми можно представить в виде макро – графа. В качестве простого примера можно привести многослойный персептрон. Задача вычисления выхода нейронной сети сводится к задаче вычисления выходов каждого из нейронов. (Рис. 1)

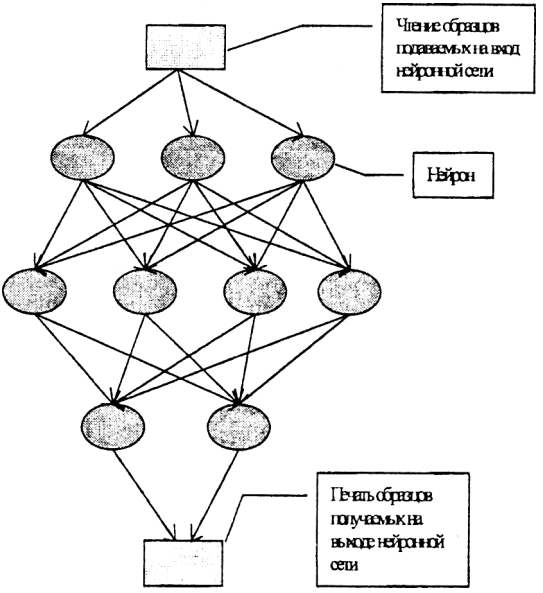


Рис. 1 (Нейронная сеть)

Описание процесса получения параллельной программы по графу алгоритма

Перед тем как получать какое-либо представление графа алгоритма. Необходимо произвести тестирование производительности процессоров и тестирование скоростей передачи данных через каналы связи многопроцессорной системы. В результате будет получен вектор производительности процессоров и набор матриц инцендентности процессоров в многопроцессорной системе со временами передачи сообщений фиксированной длинны между процессорами многопроцессорной системы. Затем собственно необходимо получить граф алгоритма.

После того как получен граф алгоритма, необходимо получить статическое расписание выполнения узлов графа алгоритма по процессорам. В процессе выполнения программы данное статическое расписание выполнения программы на многопроцессорной программе будет сломано. К нему будут обращаться, если при попытке выбрать процессор, на котором будет выполняться какой либо узел графа получилось, что для нескольких узлов графа получены одинаковые

временные характеристики. В этом случае выберется тот процессор, который указан в расписании.

В процессе выполнения параллельной программы один из процессоров резервируется под менеджер, который осуществляет управление выбором узла графа, что будет выполняться на данном процессоре. Данный менеджер осуществляет также инициализацию передачи данных по конкретному ребру графа программы.

Пользовательское описание системы

С точки зрения пользователя инструментальная система устроена следующим образом:

1. Программное средство ***graph2c++*** преобразования заданного на вход текстового представления графа алгоритма (графа зависимостей по данным, макро – граф программы) в параллельную программу с параллельными вызовами, например ***MPI,*** на языке программирования ***C++***.
2. Исходных текстов ядра системы и Make файла для совместного компилирования ядра и исходных текстов программы, полученных с помощью утилиты (***graph2c++***). Ядро управляет непосредственно процессом выполнения полученной параллельной программы. После совместной компиляции исходных файлов ядра системы с полученным представлением графа алгоритма, в виде программы написанной на языке ***C***, будет получен исполнимый файл, который можно запустить на многопроцессорной системе.
3. Утилиты ***processor_test*** и ***network_test*** для измерения производительности системы. В результате работы данных утилит должно быть получено 2 файла. Матрица со скоростями передачи данных между процессорами, а так же вектор производительности процессоров.
4. Утилиты для получения статического расписания загрузки процессоров многопроцессорной.
5. Утилиты автоматического получения графа зависимости по данным по исходному коду однопроцессорной программы. В текущий момент пользователю предлагается самостоятельно описать собственный граф алгоритма, воспользовавшись форматом текстового представления графа алгоритма. Описание в приложениях. Однако планируется использование графического интерфейса для создания и редактирования графа алгоритма. Кроме того, планируется автоматическое получение графа алгоритма по программе на языке ***C*** .

Итак, для работы системы, пользователю необходимо сформировать несколько нижеописанных файлов. При этом он может

воспользоваться соответствующими предоставляемыми системой утилитами.

1. Файл графа алгоритма (***graph.grf***) и файлы с исходными кодами узлов в графе алгоритма.
2. Расписание загрузки узлов графа алгоритма по процессорам, как желаемую последовательность выполнения на процессорах данных узлов графа алгоритма, а также желаемое прикрепление узлов графа к конкретным процессорам. (***schedule.sch***).
3. Вектор производительности процессоров многопроцессорной системы. (***procs.vct***)
4. Матрицу скоростей пересылок данных между процессорами в многопроцессорной системе. (***link.mtr***)

Последовательность действий при использовании инструментальной системы

Работа с системой разбивается на 4 этапа:

1. настройка инструментальной системы под конкретную архитектуру многопроцессорной машины;
2. подготовка задания;
3. статическая настройка задания на архитектуру;
4. Выполнение задания.

Настройка системы

1. Получить файл ***procs.vct*** с оценками производительности процессоров воспользовавшись утилитой ***processor_test.***
2. Получить файл ***link.mtr*** с оценками скоростей передачи данных между процессорами, воспользовавшись утилитой ***network_test***.

Подготовка задания

1. Подготовить файл графа алгоритма. В случае отсутствия утилиты автоматического получения графа алгоритма, пользователь системы должен вручную составить файл графа алгоритма (описание и требования к нему представлены ниже).
2. После этого, воспользовавшись программой ***graph2c++***, получить исходный файл на языке программирования ***C++,*** в котором содержатся параллельные вызовы ***MPI***. Вершины графа в полученном файле, интерпретируются как функции, где в качестве кода, подставляется текст из файлов связанных с текстовым представлением

графа алгоритма. Рёбра при этом интерпретируются как передача данных.

Статическая настройка задания на архитектуру

Получить файл ***schedule.sch,*** в котором указано на каком процессоре и в каком порядке желательно выполнение узлов графа алгоритма. Желательно, чтобы данный файл был получен автоматически, но в текущий момент такое средство не реализовано.

Выполнение задания

1. Необходимо скомпилировать полученный *C++* файл совместно с ядром системы, например, воспользовавшись стандартной утилитой ***make.*** В результате получить исполняемый файл ***prog***.
2. При помощи стандартной утилиты запуска программы на выполнение в многопроцессорной системе запустить полученную программу на выполнение. Например:

mpirun –np 8 prog graph.grf schedule.sch procs.vct link.mtr ,

указав в качестве параметров:

8 – количество процессоров;

prog – имя исполняемого файла;

graph.grf - текстовое представление графа алгоритма;

schedule.sch – файл расписания;

procs.vct - файл производительностей процессоров;

link.mtr – файл скоростей передачи данных.

Подсистема организации процесса выполнения параллельной программы.

Система состоит из узлов графа алгоритма, которые оформлены как функции с именами ***px_node_xxx***, Функции ***px_demon***, которая загружается на 0 процессор и занимается организацией порядка выполнения узлов графа, и управлением передачи данных. Функции ***main*** головной функции, которая запускается на каждом процессоре и осуществляет непосредственную загрузку узлов графа по процессорам.

Программная часть инструментальной системы основана на принципе обмена сообщениями. В системе присутствует 2 различных типа сообщений. Первый тип - управляющие сообщения, сообщения между узлами графа алгоритма. Все эти сообщения короткие, их по сравнительно немного, следовательно, когда узлы графа алгоритма

достаточно тяжёлые, управляющие сообщения не должны нагружать коммуникационную среду многопроцессорной системы. Второй тип, сообщения, посредством которых происходит передача данных между узлами графа алгоритма. Эти сообщения могут быть достаточно тяжёлыми и, как следствие, могут существенно загружать коммуникационную среду многопроцессорной системы.

В узлах графа программы, сначала, происходит приём данных, затем их обработка и передача другим узлам графа.

Устройство менеджера ресурсов (функции *px_demon*)

В функции демона входит: распределение узлов по процессорам в процессе работы параллельной программы и выбор передаваемого ребра. В функции демона входит так же поддержка синхронизации работы всей параллельной программы в целом.

Остановимся на важных функциях, необходимых для работы демона:

try_to_send – функция пытающаяся послать данные через ребро из списка не инициализированных рёбер.

to_processors – функция, распределяющая узлы из списка заявленных на выполнение на свободный процессор. В случае невозможности распределения возвращается сообщение о неудаче.

Важные структуры данных, необходимые для работы демона:

loading – массив с флагами, показывающими занятость процессора;

processors – массив с узлами, загруженными на процессоры;

waiting_recv – массив с флагами, показывающими, что процессор ждёт сведений о рёбрах для узла графа загруженного на него.

req – список заявленных на выполнение узлов;

unsent_edges – список рёбер по которым в текущий момент времени невозможно передать данные;

sending_edges – список рёбер, по которым в текущий момент производится передача данных;

Алгоритм работы демона

1. Из представления графа алгоритма в виде класса считывается слой, и узлы из него добавляются в список заявок ***req.*** Если слои закончились, то демон завершается.
2. Организуется цикл по числу заявленных на выполнение узлов графа алгоритма.

3. Если число свободных процессоров не равно 0, число незагруженных, но находящихся в списке заявок узлов равно 0 и, кроме того, текущий слой не последний, то цикл по числу заявок прерывается и к списку заявок добавляется следующий слой.

4. Выставляется ловушка, которая ловит все сообщения от процессоров, адресованные демону. На каждый вид сообщения определяется своя реакция.

5. PX_NODE_CLEAN – означает, что процессор приславший это сообщение освободился. При этом в соответствующем месте массива ***loading*** проставляется 0 и узел, который, был загружен на соответствующий процессор, удаляется из списка заявок.

6. Производится попытка распределить узлы графа, из списка, заявленных на выполнение, по процессорам. Для этого вызывается функция ***to_processor***.

7. Производится попытка инициализировать пересылку по рёбрам графа, для этого вызывается функция ***try_to_send***.

8. PX_RECIEVE_FINISH – означает, что узел пославший это сообщение принял данные от других узлов. Соответствующее ребро удаляется из ***sending_edges*** и ***unsent_edges***.

9. По приёму сообщения PX_EDGES_QUESTION в массиве ***waiting_recv*** в соответствующей позиции выставляется 1.

10. И затем для всех процессоров для которых в данном массиве выставлена 1 производится анализ наличия всех необходимых рёбер, которые требуются для работы узла загруженного на данный процессор.

11. Если все необходимые рёбра имеются, то производится передача сообщения PX_EDGES_PLACE с распределением рёбер по процессорам.

Организация выбора загружаемого узла графа на процессор

Выбор узла графа для выполнения на освободившемся процессоре из списка узлов, заявленных на исполнение производится, исходя из соображения минимизации времени выполнения данного узла графа на данном процессоре.

Каждый узел характеризуется весом. Вес показывает объём операций необходимых для выполнения узла, в эквиваленте числа операций по перемножению действительных чисел (***node_weight***).

Каждое ребро характеризуется весом. Вес показывает объём в байтах (длина сообщения) передаваемых данных. (***edge_weight***)

Производительность процессора задаётся как время выполнения эталонного числа (***num_operation***) операций с плавающей точкой. (***processor_weight***).

link_weight - Время передачи сообщения с длинной близкой к ***edge_weight.***

Среди всех узлов заявленных на выполнение производится поиск узлов с минимальным временем выполнения на процессоре, где время считается по формуле:

$$time = \frac{node\_weight \times processor\_weight}{num\_operation} + \max_{1 \le i \le N}(link\_weight(edge\_weight_i)),$$

где N число входящих рёбер. Если минимум достигается для нескольких узлов графа одновременно, то выбирается тот, который заявлен в расписании на выполнение на данном процессоре.

Описание средств тестирования производительности многопроцессорной системы

В качестве теста производительности процессоров производится одновременное перемножение 2-х случайным образом заполненных матриц с плавающими числами одинаковой размерности. Время работы при этом вычисляется в миллисекундах и исключается время простоя программы на машине. Результаты печатаются в файл как вектор с числом координат по числу процессоров.

Тест скоростей передачи данных между процессорами основан на одновременной неблокированной передаче данных от каждого к каждому. При этом необходимо указать диапазон длин сообщений, а так же шаг диапазона. Как результат получается набор матриц со временами передачи данных между процессорами для набора длин сообщений. Строки матриц задают процессор, с которого были переданы данные, а столбцы процессоры, которые принимают данные. i,j – элемент матрицы задаёт время приёма сообщения от i-го процессора j-ым. Параметры командной строки для утилиты *network_test: network_test begin_message_length end_message_length step* [*output_file*] (*network_test 10 10000 100 network.mtr*)

Формат текстового представления графа алгоритма

Инструментальная система рассчитана на двоякое представление графа алгоритма. Представление в виде класса ***Graph*** и в виде текстового файла. Приведём, формат текстового файла.

Файл графа алгоритма содержит ключевые слова, которые являются обрамляющими тегами, именами полей графа, а также их значениями. В файле графа алгоритма также могут встречаться комментарии в формате *C++*. В дальнейшем между '<' и '>' скобками будут содержаться понятия, а всё что содержится между '{' и '}' может быть повторено несколько раз. Символ ::= - означает расшифровку понятия.

Текстовое представление графа алгоритма состоит из 2-х блоков, блока узлов и блока пересылок.

<GRAPH_BEGIN>
header "*<имя файла заголовка>*"
root "*<имя файла корневого узла>*"
tail "*<имя заключительного файла>*"
num_nodes *<число узлов>*
<nodes_block>
num_edges *<число ребер>*
<edges_block>
<GRAPH_END>

Поле ***header*** - ссылка на файл, содержащий все переменные, функции, описания типов данных, которые используются узлами графа алгоритма. Следует избегать пользоваться именами, начинающимися с ***MPI*** или ***px***. Данные имена зарезервированы системой и собственно ***MPI***.

Поле ***root*** - ссылка на файл, в котором находится код, который необходимо выполнить до запуска параллельной части программы. Это необходимо обычно для проведения начальной инициализации данных, которые затем будут использоваться всеми узлами на всех процессорах. Имя файла в этом поле указывать не обязательно. Можно указать только "" вместо, например "root.grf".

Поле ***tail*** - ссылка на файл, в котором находится код, который необходимо выполнить после работы параллельной части программы.

Поле ***num_nodes*** - содержит число узлов в графе.

Поле ***num_edges*** - содержит число рёбер в графе.

<nodes_block> все узлы графа.
<nodes_block>**::=<NODES_BEGIN>**
{***<node>***}
<NODES_END>

<edges_block> все узлы графа.
<edges_block>**::=<EDGES_BEGIN>**
{***<edge>***}
<EDGES_END>

Рассмотрим теперь спецификацию каждого узла:

<*node*>::=
<NODE_BEGIN>
  number *<номер узла>*
  type *<тип узла>*
  weight *<вес узла>*
  layer *<номер слоя в графе алгоритма>*
  num_input_edges *<число рёбер входящих в данный узел>*
  edges ({*<Ребро входящее в данный узел>*})
  num_output_edges *<число рёбер выходящих из данного узла>*
  edges ({*<Ребро выходящее из данного узла>*})
  head "<имя заголовочного файла узла>"
  body "*<имя файла с кодом узла>*"
  tail "*<имя файла с заключительным кодом>*"
<NODE_END>

Поле ***number*** - уникальный номер узла в пределах всего графа. Номер узла 0 зарезервирован под корневой узел, а –1 под демон, который управляет ресурсами программы. Настоятельно не рекомендуется использовать отрицательные номера, они могут быть зарезервированы для дальнейшего расширения системной части программы.

Поле ***type*** - целочисленное значение в данный момент не используется, но в будущем возможно использование для отображения графа, и ручного выполнения этапа перестройки графа алгоритма.

Поле ***weight*** - целочисленное значение, характеризующее сложность узла должно быть выражено в неком числе эталонных операций.

Поле ***layer*** - целочисленное значение, соответствующее уровню или ярусу графа алгоритма.

Поле ***num_input_edges*** - число рёбер входящих в данный узел. (См. Рис. 1).

Поле ***num_output_edges*** - число рёбер выходящих из данного узла. (См. Рис. 1).

Поля ***edges*** - номера рёбер входящих или выходящих из данного узла. Их число должно точно соответствовать числу входящих или выходящих рёбер.

Поле ***head*** содержит имя файла с переменными, видимыми только в данном узле, а также с кодом, который необходимо выполнить до приёма данных от других узлов. Если данный файл не нужен, то в данном месте можно указать "", что будет означать отсутствие данного файла.

Поле ***body*** содержит имя файла с кодом, который необходимо выполнить, получив данные от других узлов.

Поле ***tail*** содержит имя файла с кодом, который выполнится после передачи данных.

рассмотрим спецификацию ребра:

```
<edge>::=
<EDGE_BEGIN>
 number          <номер ребра>
 weight          <вес дуги>
 type            <имя типа дуги>
 num_var         <число переменных
                  передаваемых через
                  данное ребро>
 num_send_nodes  <число посылающих узлов>
 send_nodes      ( {<номер посылающего узла>} )
 num_recv_nodes  <число принимающих узлов>
 recv_nodes      ( {<номер принимающего узла>} )
 <send_block>
 <recieve_block>
<EDGE_END>

<send_block>::=
<SEND_BEGIN>
 {<chunk>}
<SEND_END>

<recieve_block>::=
<RECIEVE_BEGIN>
 {<chunk>}
<RECIEVE_END>
```

Поле ***weight*** - целочисленное значение, характеризующее объём передаваемых данных, выраженное в числе эталонных сообщений.

Поле ***type*** задаёт тип ребра в данный момент доступно только одно значение GRAPH_NONE, которое предусматривает только передачу данных от одного процессора к единственному другому. В будущем, возможно, добавится возможность коллективно передавать данные, например GRAPH_BCAST или GRAPH_REDUCE.

Поле ***num_var*** - целочисленное значение, показывающее, количество переменных, которое будет передано через данное ребро. Все переменные упаковываются в специальный массив, а потом

передаются другому узлу, инициализируя одну пересылку, при этом на другом конце происходит распаковка из такого же аналогичного массива в собственные переменные.

Поля ***num_send_nodes*** и ***num_recv_nodes*** задают число узлов, которые принимают или отправляют данные через данное ребро. В текущей реализации, в виду отсутствия реализации коллективной передачи данных, данные поля должны быть установлены в 1.

Введём понятие – чанк (chunk). Чанк обозначает в массиве непрерывный фрагмент данных, который будет передаваться через данное ребро.

Чанки содержат информацию о передаваемых и принимаемых переменных. Один чанк – фрагмент данных связанный с одной переменной.

<chunk>::=
<CHUNK_BEGIN>
name "*<имя значения переменной>*"
type *<тип переменной>*
left_offset "*<левое смещение от адреса переменной>*"
right_offset "*<правое смещение от адреса переменной>*"
<CHUNK_END>

Поле *name* содержит строку являющуюся именем передаваемой переменной, но не её адресом в соответствующем языке программирования.

Поле *type* означает элементарный тип передаваемой переменной. (Сложные типы, например *struct* указывать нельзя.) Это должен быть тип аналогичный GRAPH_INT, GRAPH_FLOAT.

Поля *left_offset* и *right_offset* задают смещение в элементах данного типа относительно начала, т.е. адреса переменной. Отрезок данных между *left_offset* и *right_offset* передаётся другому узлу графа алгоритма. Значения данных полей – целочисленные выражения в терминах того языка программирования, на котором предполагается последующее получение параллельной программы.

Файл расписания устроен аналогично файлу графа алгоритма, единственно, номера процессоров должны начинаться с первого. Нулевой процессор зарезервирован за демоном, осуществляющим взаимодействие с узлами графа.

<schedule>::=
<SCHEDULE_BEGIN>
num_processors *<Число процессоров>*
{***<processor>***}
<SCHEDULE_END>

<processor>::=
<PROCESSOR_BEGIN>
name *<номер процессора>*
num_nodes *<число узлов распределённое на процессор>*
nodes ({*<номер узла>*})
<PROCESSOR_END>

Литература

1. В.В. Воеводин "Информационная структура алгоритмов." Издательство Московского университета 1997г.
2. А.Б. Барский «Параллельные процессы в вычислительных системах. Планирование и организация». М. Радио и связь 1990.
3. Всероссийская научная конференция "Фундаментальные и прикладные аспекты разработки больших распределённых программных комплексов." – "Система крупноблочного распараллеливания Parallax2"
4. Michael J. Quinn "Efficient Algorithm for parallel Computers"
5. Ян Фостер "Design and Building Parallel Programs".
6. www.parallel.ru
7. www.netlib.org
8. www.anl.gov
9. International Symposium on Parallel Architectures, Algorithms and Networks (December 1994.) "Increase Analysis in the Total Execution Time of a Parallel Program."
10. International Symposium on Parallel Architectures, Algorithms and Networks (December 1994.) "A Detailed Analysis of A Random Polling Dynamic Load Balancing."
11. International Symposium on Parallel Architectures, Algorithms and Networks (December 1994.) "A Greedy Task Clustering Heuristic That is Provably Good."
12. International Symposium on Parallel Architectures, Algorithms and Networks (December 1994.) "Strategy and Simulation of Adaptive RID for Distributed Dynamic Load Balancing in Parallel System."
13. First International Conference on Massively Parallel Computing Systems "Dynamic Load Distribution in Massively Parallel Architectures: the Parallel Objects Example."
14. First International Conference on Massively Parallel Computing Systems "Automatic Load Balancing in Parallel Direct Simulation Methods."
15. First International Conference on Massively Parallel Computing Systems "Automatic Load Balancing in Parallel Direct Simulation Methods."

William Gropp, Ewing Lusk, and Anthony Skjellum. "Using MPI"

Вдовикин О.И., Машечкин И.В.

О проблеме построения параллельной файловой системы для кластерных ВС

Введение

В настоящее время всё более широкое распространение получают кластерные высокопроизводительные вычислительные системы, имеющие в своём составе до нескольких сотен, а иногда и тысяч узлов, каждый из которых работает под управлением собственной операционной системы. В процессе эксплуатации таких систем возникает проблема эффективной организации конкурентного параллельного доступа к файлам. Как правило, такая задача возлагается на параллельные файловые системы.

В данной работе рассматриваются общие подходы, используемые при построении параллельных файловых систем для многопроцессорных вычислительных комплексов, и предлагается вариант построения распределённой параллельной файловой системы, специфично нацеленной на использования в типичных вычислительных кластерах.

Как правило, структура такого кластера может быть представлена в виде совокупности нескольких компонент:

1. вычислительные модули, входящие в состав решающего поля;
2. вычислительная сеть, объединяющая ресурсы решающего поля;
3. сеть управления и устройства внешней памяти.

Рисунок 1 иллюстрирует такую структуру.

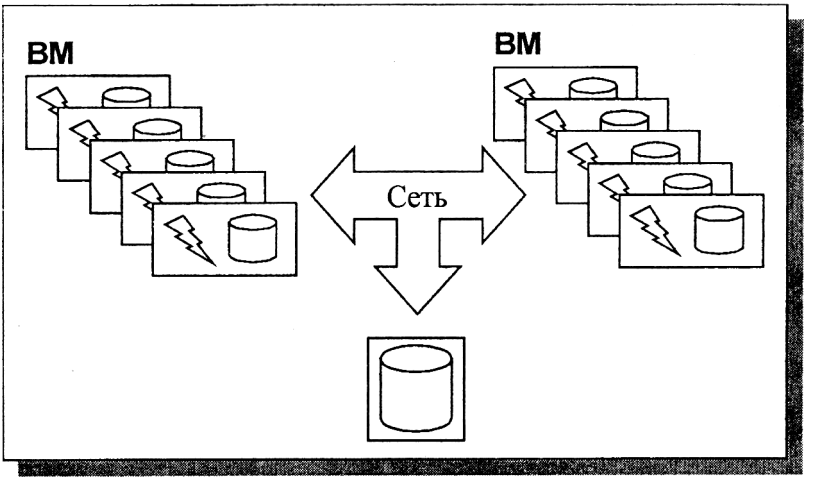


Рисунок 1. Типичная структура кластерной системы

Параллельный ввод-вывод

Параллельный ввод-вывод - высокоуровневый интерфейс, поддерживающий разбиение файловых данных между процессами, а также коллективные операции поддерживающие сложные пересылки структур данных между памятью процессов и файлами. При этом в качестве иллюстрации структуры «параллельного» файла может служить подход использующийся в библиотеках MPI-IO [2]. В качестве основного хранимого в параллельном файле типа данных выступает один из элементарных типов MPI [1]. Дополнительно каждая из ветвей параллельной программы определяет свой собственный тип данных - файловый тип данных - образованный либо из элементарного типа, либо из отнаследованного от него. При этом, файловый тип данных задаётся для каждой ветви отдельно, и в совокупности они формируют «запись» в параллельном файле. Рисунок 2 иллюстрирует этот подход.

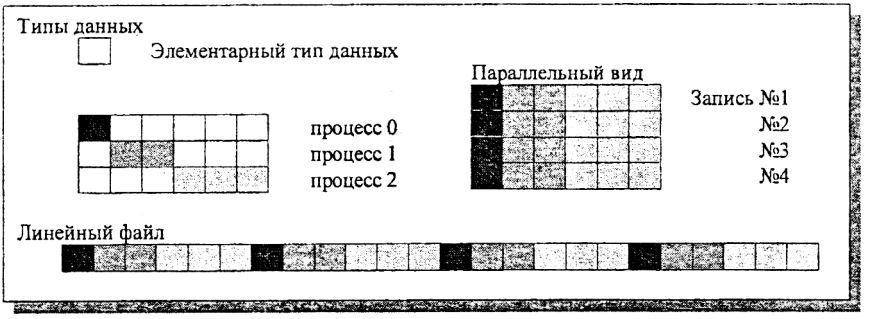


Рисунок 2. Типы данных и представления файлов в MPI-IO

При такой организации «параллельный» файл может быть представлен как последовательный, состоящий из отдельных записей. В дальнейшем все манипуляции над файлом производятся, как с линейным файлом. Т.е. файловая система при его хранении более не учитывает его структуру. Такой подход применяется в большом количестве параллельных файловых систем. Среди них PVFS [5], разработанная для кластерных вычислительных систем. В параллельных системах такого рода различают как минимум два типа узлов - узлы ввода-вывода и вычислительные. Первые ответственны за формирование параллельной файловой системы, вторые - пользуются их сервисом. Как правило, эти два множества не пересекаются. Такой же подход к организации ввода-вывода обычно используется и при построении некластерных мультипроцессорных систем. Рисунок 3 представляет такую классическую схему.

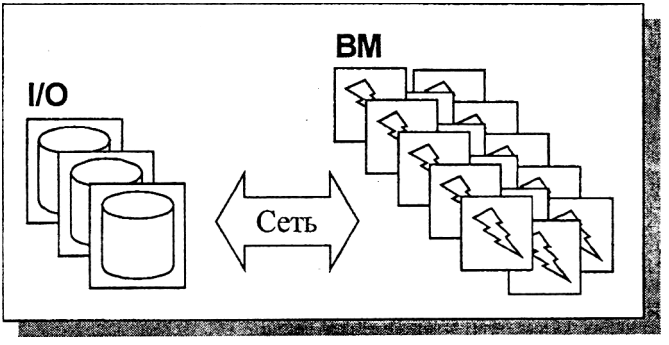


Рисунок 3. Классическая схема организации ввод-вывод

При использовании такого подхода для хранения линейного файла обычно применяется приём, называемый stripping - разбиение

файла на части фиксированного размера и «наматывание» этого файла на узлы-ввода вывода.

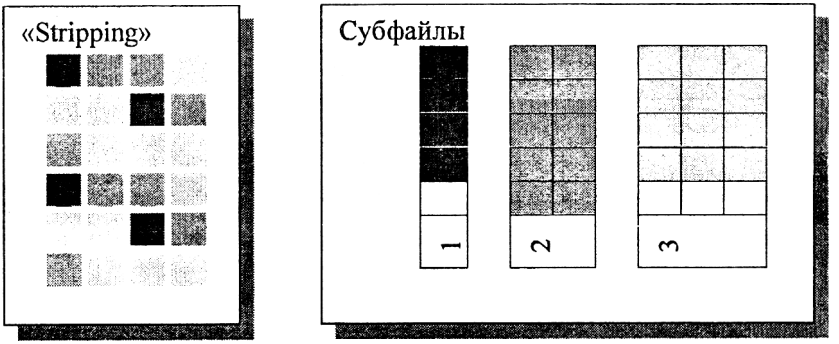


Рисунок 4. Stripping и субфайлы

При этом для доступа к файлу может строиться запрос одновременно к нескольким узлам ввода-вывода. Это ведёт к проблемам синхронизации при доступе к параллельному файлу, поддержании консистентности кэшей, файловых указателей и т.п. При таком подходе проблематично учесть и разную производительность, участвующих в вычислении узлов: поскольку файл линеен, то запись в него происходит последовательно и т.о. быстрые узлы вынуждены дожидаться соседей.

Другой подход, призванный решить такую проблему - разбиение файла на субфайлы. При этом параллельный файл представляется как набор отдельных файлов (субфайлов), каждый из которых может обрабатываться независимо. Такой подход, например, используется в Vesta [3] и созданной на её основе PIOFS [4]. Эти файловые системы используют классическую схему организации ввода-вывода, рассмотренную ранее, и stripping данных, но уже не на уровне файлов, а на уровне субфайлов. При использовании такого подхода резко упрощаются задачи по синхронизации ветвей параллельной программы. В большинстве случае она становится просто не нужна [3].

Параллельная файловая система для кластеров

Концепция

В различных областях прикладных вычислений существует большой класс параллельных задач, использующих файловые системы

таким образом, что основной обмен данными между ветвями параллельной задачи осуществляется не через файлы, а с помощью прикладных библиотек, таких как MPI. При этом файлы используются в основном как средство для хранения входных и выходных данных.

Такими свойствами обладают, например, большинство задач с визуальными средствами пост-обработки. При этом большая часть таких задач логическим образом разбивают результирующие данные по числу участвующих в счёте ветвей и каждая их них независимо использует либо индивидуальный файл, либо часть общего параллельного файла. Другим ярким примером такого рода задач являются, например, вычисления с сохранением контрольных точек. В этом случае каждая из участвующих в вычислении ветвей периодически сохраняет в файле свои текущие данные.

Естественным образом такие задачи приходят к понятию субфайла и отображений одного параллельного файла на все участвующие в счёте процессы.

Предположение о локальности обменов процессов в кластерной вычислительной системе легло в основу проектирования предлагаемой файловой системы; оно же явилось прямой предпосылкой использования устройств ввода-вывода, имеющихся непосредственно в вычислительных узлах. При этом прикладное программное обеспечение строится таким образом, чтобы максимально увеличить вероятность локальности обмена.

По сути, основная проблема большинства существующих параллельных файловых систем, при использовании их подобными задачами - попытка хранения параллельных данных в последовательном виде, с дальнейшей их реконструкцией в параллельный.

Общая структура системы

В предлагаемой параллельной системе комбинируются подходы, используемые в PVFS и Vesta: во-первых, для физического хранения файлов используются локальные диски узлов вычислительного кластера, во-вторых, используется разбиение файла на субфайлы. При этом, в отличии от обоих систем, не выделяются отдельно узлы ввода-вывода и вычислительные - основная идея системы состоит именно в возможности (и необходимости) использования вычислительных узлов для выполнения операций с файлами.

Структура хранения данных тоже имеет существенные отличия от рассмотренных систем. В PVFS осуществляется разбиение (stripping) последовательного файла на фрагменты, и дальнейшее его хранение на

наборе различных узлов ввода-вывода. В Vesta, оперирующей субфайлами, тоже происходит подобный процесс. При этом в обеих системах хранимые на узлах ввода-вывода данные сами по себе не несут никакой ценности - т.е. для получения целостных данных необходимо «собрать» информацию с группы узлов. В предлагаемой системе, весь субфайл (т.е. логически организованная структура) хранится целиком только на одном узле ввода-вывода, т.е. дальнейшее разбиение данных внутри одного субфайла не производится. Поскольку в данном случае субфайл - это, по сути, обычный файл, появляется возможность отказаться от разработки и использования специальной локальной файловой системы для хранения данных на узлах ввода-вывода. Более того, такая структура позволяет упростить не только хранение данных, но и массу связанных с этим задач, например их резервное копирование.

Другое немаловажное отличие - уровень, на котором обеспечивается параллелизм в файловой системе. Существующие системы параллельны «внутри», т.е. оперируют файловой системой как единым целым, таким образом метаданные (аналогичные суперблоку, индексным дескрипторам, спискам свободных блоков и т.п. локальной файловой системы) «размазываются» по всему хранилищу файловой системы. К сожалению, данный подход может быть использован только при условии 100% устойчивости системы к сбоям как аппаратуры, так и программного обеспечения: разрушение части данных только на одном узле ввода/вывода может привести к краху всей файловой системы. В условиях использования кластерной системы, состоящей из сотен узлов, достичь такой стабильности невозможно. Поэтому, предлагается обеспечить параллелизм не на уровне средств файловой системы, а на уровне библиотек, например таких как MPI-IO. При этом в качестве базовой файловой системы разработать и использовать распределённую файловую систему со средствами поддержки параллельных прикладных библиотек.

Организация данных

Рисунок 5 представляет общую структуру предлагаемой параллельной файловой системы. Логически файловое пространство разбивается на три части:

1. дисковое пространство, доступное вычислительной задаче для хранения и обработки данных локально;
2. пространство, доступное вычислительной задаче для хранения и обработки данных и расположенное на других узлах вычислительного кластера;

3. метаданные, используемые для организации «параллельности» в файловой системе и используемые прикладной библиотекой.

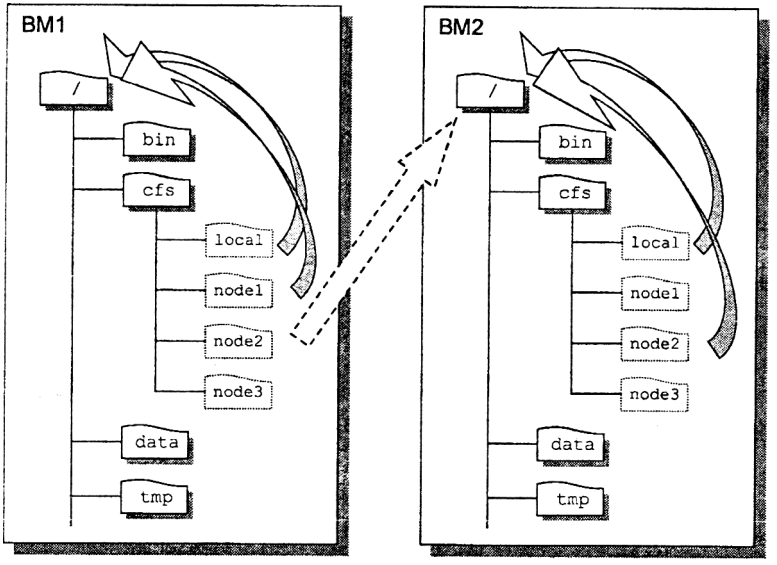


Рисунок 5. Организация данных в системе

Для обеспечения прозрачности именования в рамках вычислительной системы используется выделенная и заранее известная точка «монтирования» файловой системы. В данном случае это каталог /cfs. Рисунок 5 поясняет структуру первых двух частей системы.

Точка монтирования /cfs - корневая вершина распределённой файловой системы. Все её непосредственные потомки синтезируются программным обеспечением. Для их именования используются имена узлов, входящих в вычислительную систему. На рисунке это node1, node2 и node3. Кроме этого существует специальная вершина local, обозначающая часть распределённой системы, данные которой хранятся локально.

Программное обеспечение построено таким образом, что с точки зрения операционной системы каталог local и каталог с именем, совпадающим с именем хоста, являются по сути символическими ссылками на корневую вершину / локального дерева каталогов. Остальные непосредственные потомки /cfs представляются каталогами. Т.о. после разрешения соответствия имени индексному дескриптору файла, все дальнейшие обмены с файлом, расположенным локально, идут с использованием «обычных» операций над локальными файловыми системами. Для работы с файлами, расположенными не на

локальном узле, используется программное обеспечение распределённой файловой системы.

Важно отметить, что, с точки зрения операционной системы, «смонтированной» является только корневая вершина /cfs (альтернативой могла бы стать версия, когда «монтируются» только нелокальные каталоги, а остальные представлены символическими ссылками; однако такое решение привело бы к необходимости «монтирования» к сотням точек, что используемая операционная система не поддерживает), все остальные задачи по представлению и трансляции структуры каталогов возлагаются на разрабатываемое программное обеспечение.

Следующим уровнем, который позволяет организовать «параллелизм» в файловой системе, является хранилище метаданных. В качестве такого хранилища предлагается использовать сетевую файловую систему NFS, как наиболее стабильную и простую в использовании. Рисунок 6 поясняет структуру метаданных в параллельной файловой системе.

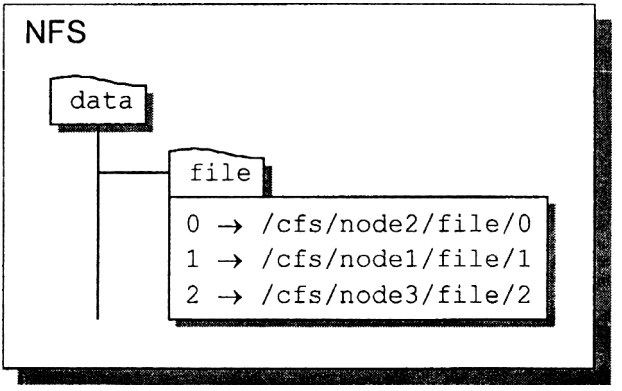


Рисунок 6. Метаданные в параллельной системе

Структура параллельного файла в метаданных представляется как набор субфайлов. При этом параллельному файлу в общей структуре хранения метаданных соответствует каталог, а субфайлам - содержимое этого каталога. Поскольку данные субфайлов хранятся не на NFS-сервере, а на вычислительных узлах кластера, вместо реальных файлов здесь хранятся только символические ссылки, указывающие реальное местоположение файла в распределённой файловой системе.

По сути, метаданные позволяют осуществить разбиение файла на субфайлы и указать прикладным библиотекам и программам места хранения этих субфайлов.

Программные компоненты системы

Начальная реализация и перспективы

Программное обеспечение, формирующее предлагаемую параллельную файловую систему, разбивается на четыре части:

1. «виртуальная» файловая система, функционирующая как часть операционной системы в привилегированном режиме, обеспечивающая формирование непосредственных потомков вершины /cfs и осуществляющая диспетчеризацию обращений к ним;
2. сервер доступа к локальной файловой системе, обрабатывающий запросы удалённых клиентов;
3. клиенты удалённых файловых систем;
4. прикладные библиотеки параллельного ввода/вывода, формирующие «образ» файловой системы, как параллельной.

Поскольку многопроцессорные вычислительные кластеры, как правило, используются несколькими задачами одновременно, следует обратить особое внимание на создание наиболее «легковесной» серверной части: в случае, если задача обращается за данными, расположенными на вычислительном узле, в данный момент используемом другой параллельной задачей, влияние сервера доступа на производительность узла должна быть минимальной. Именно поэтому компонента сервера доступа должна работать в режиме ядра операционной системы, с тем чтобы уменьшить количество переключений контекстов, использовать лишь буферный кэш оперативной памяти, осуществлять только асинхронную обработку дынных и т.п. Наибольшую нагрузку в этом случае должен нести клиент доступа, именно он должен, например, заниматься разбором имён.

С этой точки зрения, целесообразно в начальной реализации использовать наработки, имеющиеся в операционной системе для поддержки работы NFS-сервера. Семантика NFS-операций наилучшим образом соответствует выдвигаемым требованиям. Особую роль в этом играет изначальная концепция, заложенная в протокол NFS - это сервер без состояния (state-less) и использование вызова удалённых процедур (RPC) для осуществления обменов.

Особенностью NFS-модели является и то, что при использовании данного протокола, в качестве транспорта сообщений может выступать сервис с негарантированной доставкой. Это позволяет, с одной стороны, не занимать сетевые каналы пакетами подтверждений, а с другой - максимально разгрузить NFS-сервер. Нормальное функционирование данной модели обусловлено тем, что клиент может

повторять все операции, на которые он не получил подтверждения об исполнении, любое количество раз, при этом даже повторное их исполнение не приводит к каким-либо проблемам.

Дальнейшее развитие клиент-серверных программных компонент системы может пойти по следующим направлениям:

1. использование библиотеки Nexus Globus Toolkit [8], с возможностью дальнейшего использования в Grid-системах;
2. использование технологии Active Messages [9] для передачи сообщений и ещё большей разгрузки сервера;
3. разработка собственного протокола взаимодействия, аналогичного по семантике NFS, но допускающего использование нескольких каналов передачи данных (сетей в кластерных системах) для транспортировки запросов на исполнение операций, их ответов и данных.

Прикладная библиотека параллельного ввода/вывода

Предполагается, что в качестве основной библиотеки параллельного ввода-вывода в параллельных программах будет использоваться библиотека MPI-IO. При этом, основной упор будет нацелен не на получение собственного набора параллельных программных интерфейсов, которые потом будут использоваться для реализация набора функций MPI-IO, а на реализацию только набора интерфейсов ADIO [7], используемых в популярной реализации MPI MPICH [6], доступной для использования на большинстве кластерных вычислительных систем.

Одна из задач предлагаемой реализации - формирование и поддержание метаданных системы, поскольку именно они определяют «параллелизм» в файловой системе.

Обеспечение максимальной локальности обменов и поддержание метаданных осуществляется в реализации вызова ADIO_Open. При открытии нового параллельного файла на запись, происходит формирование всех необходимых субфайлов для параллельных ветвей программы по следующему сценарию:

1. переданная в качестве имени параллельного файла строка будет использоваться для создания одноимённого каталога в области метаданных;
2. ветви параллельной программы, будучи запущенными на нескольких узлах кластера, создают одноимённые каталоги в корневой вершине распределённой файловой системы /cfs/local;

3. в этих каталогах, в области данных, создаются регулярные файлы с именами, соответствующими номеру ветви параллельной программы;

4. в каталоге, используемом для хранения метаданных, создаются символические ссылки на файлы в распределённой файловой системе.

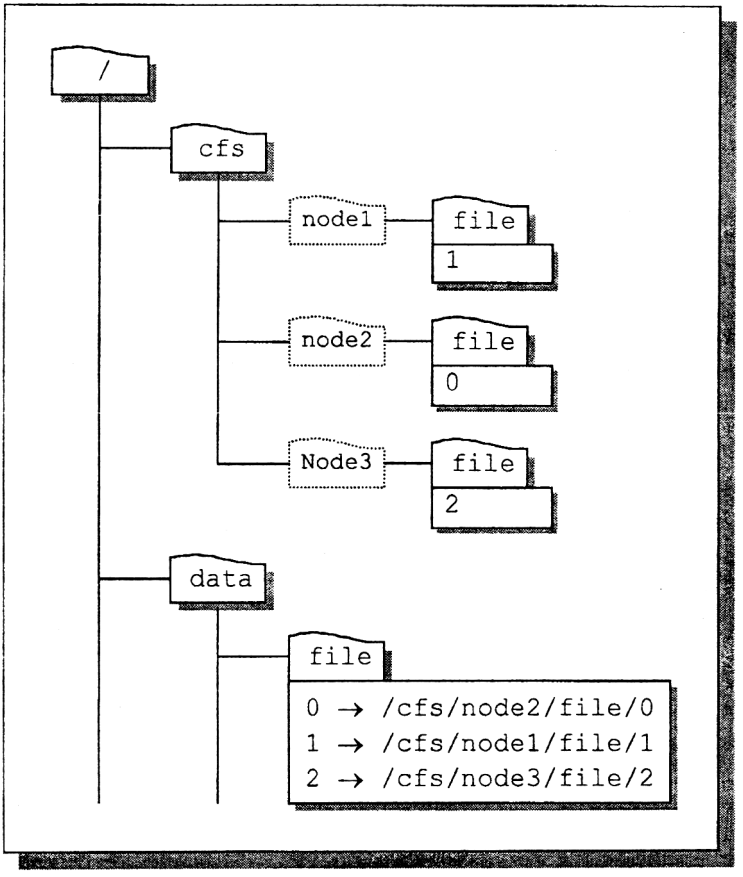


Рисунок 7. Связь метаданных и субфайлов

Рисунок 7 иллюстрирует состояние областей метаданных после проведения такой операции при открытии файла с именем file тремя параллельными ветвями одной задачи, работающими на трёх различных узлах: node2 для ветви 0, node1 для ветви 1 и node 3 для ветви 2.

Аналогичным образом разрешаются ссылки и при открытии файлов на чтение. Следует, однако, заметить, что при большом объёме

входных данных может оказаться выгодным введение дополнительного «нулевого» шага, на котором ещё до запуска задачи осуществляется копирование необходимых файлов на узлы кластера, выделенные для счёта данной задаче, с соответствующей корректировкой метаданных.

Отметим, что предложенная организация параллельной файловой системы позволяет использовать её возможности и программам, не использующим MPI-IO. Обеспечивается это тем, что субфайлы представляют собой регулярные, последовательные файлы, доступ к которым возможен с использованием стандартных системных вызовов. Эта же особенность используется при реализации операций чтения/записи данных в ADIO.

Обеспечение отказоустойчивости, восстановление после сбоев

Существенным требованием к параллельным файловым системам является устойчивость системы к сбоям, возможность резервного копирования данных и возможность их восстановления. В большинстве существующих реализаций выход из строя одного из узлов ввода-вывода или его диска может привести либо к потери части данных, либо, в случае разрушения метаданных - к потери всех данных (обычно структура метаданных описывает расположение и способ разбиения файла на блоки; без этой информации мы имеем просто набор байт). Восстановление после сбоя в таких системах практически невозможно, поскольку повреждённый диск может хранить информацию всех файлов. Как правило в этом случае перестраивают всю файловую систему целиком. В связи со спецификой последовательно-параллельных файлов затруднено и их резервное копирование, поскольку способы их хранения далеко не просты. И если вынести их из системы в последовательном виде можно, то вернуть обратно - практически невозможно.

В предлагаемой системе могут быть выделены два основных класса отказов:

1. повреждение или полное разрушение метаданных системы;
2. повреждение или полное разрушение субфайлов.

В первом случае, возвращение файловой системы в исходное состояние требует лишь простой операции перестройки метаданных - это может быть легко сделано, с учётом того, как система оперирует с файлами и субфайлами. Т.е. в предлагаемой системе отказ сервера метаданных не влечёт к потери всех данных.

Во-втором случае, в зависимости от структуры данных разрушение субфайла может вести как к полной потери данных, так и

лишь к частичной. Однако следует заметить, что поскольку субфайлы - это регулярные файлы, то к ним применимы стандартные средства резервного копирования и восстановления, что резко упрощает задачу поддержки файловой системы.

Литература

1. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. Version 1.1, Июнь 1995.

2. The MPI-IO Committee. MPI-IO: A Parallel File I/O Interface for MPI, Version 0.5. World Wide Web <http://lovelace.nas.nasa.gov/MPI-IO>, Апрель 1996.

3. P. F. Corbet, D. G. Feitelson. The Vesta Parallel File System. ACM Transactions on Computer Systems, 14 (3): 225-264 (1996)

4. P. F. Corbet, D. G. Feitelson, J-P. Prost, G.S. Almasi, S.J. Baylor, A.S. Bolmarich, Y. Hsu, J. Satran, M. Snir, R. Colao, B.D. Herr, J. Kavaky, T.R. Morgan, and A. Zlotek. Parallel file systems for the IBM SP computers. IBM Syst. J. 34(2), с. 222-248, 1995

5. P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A Parallel File System For Linux Clusters. Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, Октябрь 2000, с. 317-327

6. W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High Performance, Portable Implementation of the MPI Message-Passing Interface Standard. Parallel Computing, 22(6):789-828, Сентябрь 1996.

7. R. Thakur, W. Gropp, E. Lusk. An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces. Proceedings Of the 6th Symposium on the Frontiers of Massively Parallel Computation, Октябрь 1996, с. 180-187.

8. The Globus Project. The Nexus Multithreaded Communication Library. World Wide Web http://www.globus.org/nexus/

10. D. Culler, K. Keeton, L.T. Liu, A. Mainwaring, R. Martin, S. Rodrigues, K. Wright, C. Yoshikawa. The Generic Active Message Interface Specification. White Paper, 1994 (World Wide Web http://now.cs.berkeley.edu/Papers/Papers/gam_spec.ps)

Раздел V
Сообщения

Леонов М.В., Глазов А.Э.

ИПС по персоналиям ботаников с доступом через Интернет

Введение

В статье представлена разработанная в лаборатории вычислительного практикума и информационных систем факультета ВМиК МГУ информационно-поисковая система с базой данных по авторам таксонов растений семейства Зонтичных. При разработке системы мы преследовали две цели. Первая – предоставить коллегам-ботаникам (см. [1]) полезный инструмент для научной работы, вторая – на примере решения достаточно содержательной, но не громоздкой задачи подготовить учебный материал для изучения студентами вечернего отделения факультета ВМиК языка SQL и других вопросов, связанных с разработкой и программированием баз данных.

Актуальность задачи и выбор инструментария

Одной из особенностей таксономической ботаники является важность корректного написания названий таксонов растений, которые, начиная со времен Карла Линнея, содержат по определенным правилам записываемую так называемую авторскую комбинацию - сочетание сокращений фамилий авторов, впервые опубликовавших или пересмотревших данный таксон.

Ситуация осложнена тем, что в докомпьютерное время, да иногда и сейчас, в разных работах используются разные транслитерации фамилий ботаников, а также различные их сокращения. Отсюда ясно, насколько полезна может быть ИПС, которая позволила бы быстро получать ответ о рекомендуемом сокращении фамилии, либо проверить корректность транслитерации всей фамилии. Известная нам аналогичная система по персоналиям ботаников (http://www.ipni.org), входящая в International Plant Names Index (IPNI), созданный в сотрудничестве ведущих ботанических институтов (The Royal Botanic Gardens, Kew, The Harvard University Herbaria, Australian National

Herbarium) обладает существенным недостатком - нет устаревших сокращений и различных транслитераций фамилий - она содержит «рекомендуемые» сокращения, и, естественно, недоступна для наших изменений и неудобна для некоторых применений. Нет в ней и возможности получить список авторов по какому-нибудь содержательному критерию. Аналогичными недостатками обладает система Index Kewensis 2.0 на CDROM, на которую имеет авторское право Oxford University Press. Правда, нашу систему мы предполагаем использовать для инвентаризации сокращений и фамилий не всех ботаников, а лишь исследователей семейства Зонтичных (в частности, для генерации соответствующего справочника).

Выбор программных средств создания базы данных, как известно, дело не простое. Приходится учитывать вопросы совместимости с уже используемыми в коллективе БД, перспективы модернизации и т.д. Совсем недавно в ботанических коллективах при создании баз данных преобладало использование СУБД семейства xBase. С натиском системы Windows увеличилось количество настольных ИС, изготовленных с помощью СУБД Access. (См., например, [2]). По мере освоения возможностей Интернет, все чаще появляются ИС, изготовленные с помощью СУБД на основе технологии «клиент-сервер», и языка SQL в том или ином варианте, в частности СУБД MySql. Учитывая то, что доступ в Интернет пока к сожалению, не является непременным атрибутом компьютера, за которым трудится наш ботаник, мы разработали проект ИПС, которая могла бы существовать на двух платформах, в частности, работать и на локальной машине без доступа к Интернет, но обладать тем же современным интерфейсом.

Поэтому в качестве программных средств были выбраны бесплатно распространяемые WWW-сервер Apache, СУБД Mysql и язык PHP. Так как перечисленные популярные продукты имеют версии и для UNIX и для Windows, обеспечение двухплатформенности не представляет особых сложностей. Отметим и относительно скромные требования указанных программных средств к дисковому пространству – версии Apache 1.3.26, MySQL 3.23.52 и PHP 4.05 вместе занимают менее 40 Мб.

Принцип построения системы и взаимодействия с пользователем

Наша ИПС может использоваться как автономно, так и внутри сети. При этом сетевой доступ возможен не только из локальной сети, окружающей сервер ИПС, но и из любой точки глобальной сети Internet.

То есть пользователь услуг ИПС может получать требуемую информацию, а администратор ИПС и редактор БД может вносить изменения в систему через Интернет.

Взаимодействие пользователя с базой данных осуществляется в соответствие со схемой, приведенной на рисунке. Эта схема применима независимо от того, как работает пользователь - удаленно или локально.

Указанные три элемента (Apache, Mysql и PHP) используются исключительно на машине-сервере ИПС. Для практического использования, то есть для внесения и извлечения информации, необходим браузер. Естественно браузер должен быть установлен как на машине-сервере ИПС, так и на всех удаленных машинах пользователей, желающих иметь дело с этой ИПС.

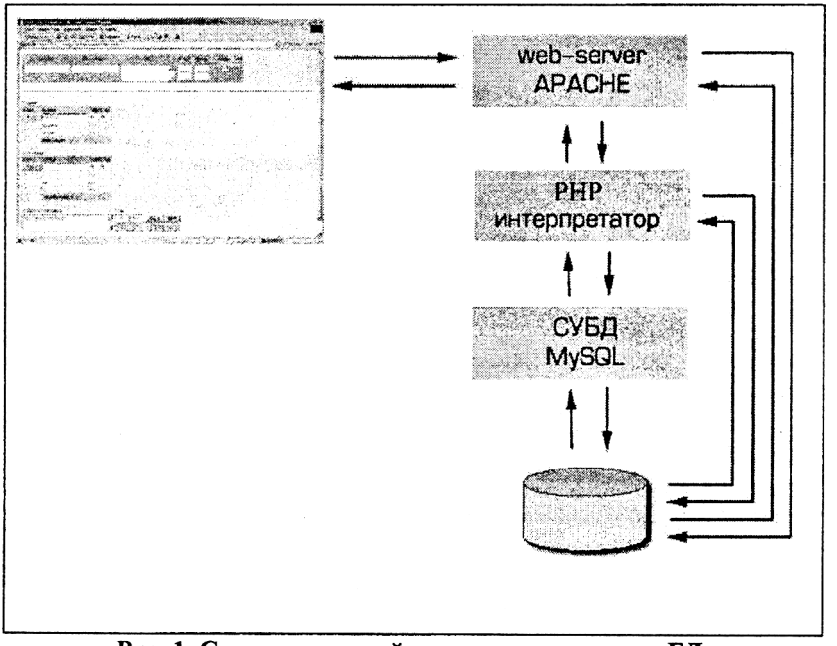


Рис. 1. Схема взаимодействия пользователя с БД.

Главная стартовая страница (см. рис. 2), а также стартовые страницы для режима пользователя и режима редактора базы написаны на языке HTML. Скрипты, реализующие пункты меню, написаны на PHP. Предусмотрены три интерактивных режима: режим администрирования, в котором можно управлять пользователями (заводить новых, исключать) посредством соответствующих команд серверу MySQL, режим простого пользователя, которому разрешено

осуществлять поиск и получать сводки, и режим привилегированного пользователя (редактора базы), который может вводить новые данные и исправлять имеющиеся. В последнем режиме предусмотрен так называемый пакетный ввод, при котором в базу может быть введено большое количество данных из специальным образом подготовленного файла, что весьма удобно при начальном наполнении базы, либо ее восстановлении.

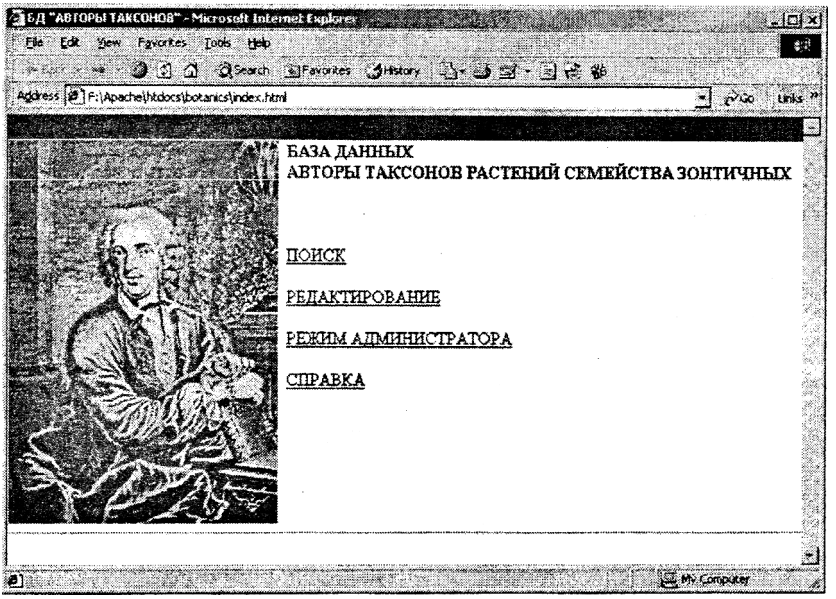


Рис.2. Главная страница системы.

Разграничение доступа с помощью пароля предусмотрено только для сетевого использования ИПС. Наличие или отсутствие парольной защиты определяется с помощью переключателя, устанавливаемого в файле настройки для модуля авторизации. Для локального использования ИПС модуль отключен, чтобы не усложнять интерфейс рядового пользователя, так как удобство пользователя-ботаника в данном случае, на наш взгляд, важнее безопасности.

В систему включена утилита, которую можно условно назвать "клиентский web-интерфейс к серверу базы данных". Он реализован с помощью двух файлов sql.php и sql_settings.php. Этот интерфейс позволяет в окне браузера вводить любые SQL-запросы, понятные серверу MySQL. Очевидно, что работа с командной строкой сложнее и требует предварительной подготовки, но в конечном итоге дает больше возможностей. Таким образом человек, знакомый с синтаксисом SQL, может формулировать запросы любого вида и получать результат, которого ему не удалось бы достигнуть, используя лишь графический интерфейс. Эта утилита предназначена для администратора, либо разработчика, сопровождающего систему.

Структура базы данных

Интересно, что логическая структура данных рассматриваемой задачи, несмотря на простоту, типична для многих других задач таксономической ботаники, так как и здесь присутствует так называемая «синонимия» названий – когда одному объекту может соответствовать несколько названий, одно из которых на данный момент является принятым. Причем надо иметь ввиду, что статус названия может измениться – название, которое было принятым, впоследствии может стать «устаревшим», и наоборот. К нашей БД в соответствии с условиями поставленной задачи были предъявлены следующие требования:

- Для каждой личности должно хранится написание фамилии (surname_spelling), а также признак того, является ли это написание принятым или нет (surname_validity) . У каждой личности может быть несколько написаний фамилий (одно или больше).
- Аналогично для каждой личности должно хранится написание сокращения (abbreviation_spelling), а также признак того, является ли это сокращение принятым или нет(abbreviation_ validity). У каждой личности может быть несколько написаний сокращений (одно или больше).
- Для каждой личности должен храниться год рождения (birth_year). Если он неизвестен, то должно быть проставлено пустое значение.
- Для каждой личности должен хранится год смерти (death_year). Если он неизвестен или отсутствует, то должно быть проставлено пустое значение.
- Также должна быть возможность хранить некоторую информацию произвольного вида (person_info). Ее

впоследствии предполагается использовать для генерации справочника.

Пример ненормализованных данных для этой базы приведен в следующей таблице:

I	Фамилии	признa	Сокращения	признa	Дополнительная информация	Год рождения	Год смерти
1	Linnaeus Linney Linnei	0 1 1	L. Lin	0 1	...	1707	1778
2	Smith	0	Sm. S.	0 1	NULL	1759	1828

Переписав эти данные в 1НФ, получим следующую таблицу:

Person_id	Surname_id	Surname	Surname_validity	Abbreviation_id	Abbreviation	Abbreviation_validity	Person_info	Birth_year	Death_year
1	1	Linnaeus	0	1	L.	0	...	1707	1778
1	1	Linnaeus	0	2	Lin	1	...	1707	1778
1	2	Linney	1	1	L.	0	...	1707	1778
1	2	Linney	1	2	Lin	1	...	1707	1778
1	3	Linnei	1	1	L	0	...	1707	1778
1	3	Linnei	1	2	Lin	1	...	1707	1778
2	4	Smith	0	3	Sm.	0	NULL	1759	1828
2	4	Smith	0	4	S.	1	NULL	1759	1828

Опуская промежуточные этапы преобразования исходного отношения, приведем окончательный набор отношений нашей БД. Это три отношения P, S и A:

P={person_id, person_info, birth_year, death_year}

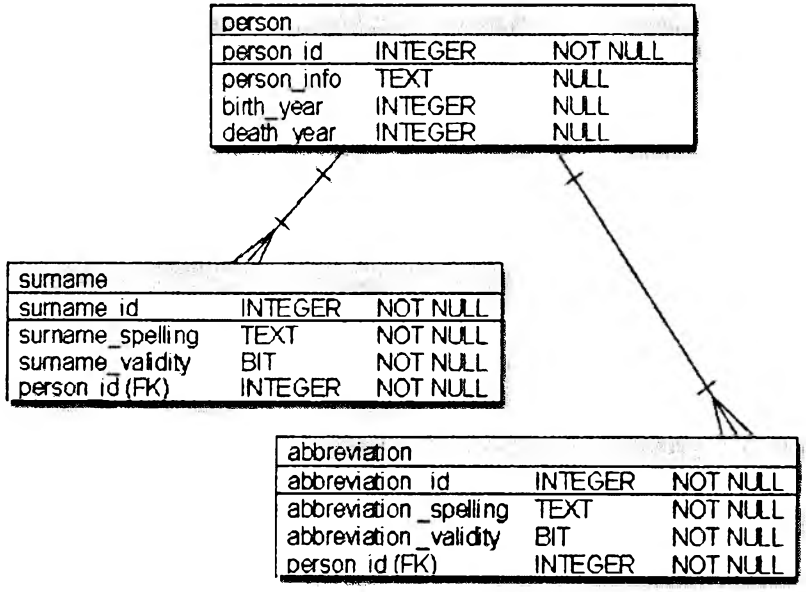
S={surname_id, surname_spelling, surname_validity, person_id}

A={abbreviation_id, abbreviation_spelling, abbreviation_validity, person_id}

Отношение	Первичный ключ
P	Person_id
S	Surname_id
A	Abbreviation_id

Для каждого из отношений указанный первичный ключ является единственным потенциальным ключом. При этом в отношении S атрибут person_id является внешним ключом для отношения P, в отношении A атрибут person_id также является внешним ключом для отношения P. Заметим, что каждое из отношений находится в 4НФ.

Такое разделение вполне естественно, так как в данной задаче можно выделить три сущности – личность, фамилия и сокращение, и теперь в одной таблице хранятся данные только о личностях, в другой – только о фамилиях, в третьей – только о сокращениях. Это может быть отражено с помощью следующей диаграммы.



На диаграмме изображены три сущности: person, surname, abbreviation. Сущность Person имеет атрибуты person_id, person_info, birth_year, death_year. Сущность surname имеет атрибуты surname_id,

surname_spelling, surname_validity. Сущность Abbreviation имеет атрибуты abbreviation_id, abbreviation_spelling, abbreviation_validity.

Связь между сущностями person и surname, а также между person и abbreviation есть связь типа один-ко-многим, причем наличие экземпляров сущности с каждой стороны является обязательным.

Организация интерфейса пользователей

Работа редактора базы

На стартовой странице содержится форма для ввода информации. В раскрывающемся списке можно выбрать одно из действий, основными из которых являются «простой поиск» и «добавить личность». Выбор всех пунктов за исключением пункта "добавить личность" подразумевает тот или иной вид поиска. При этом необходимо указать критерии поиска, заполнив соответствующие поля формы: "Фамилия", "Сокращение", "Информация", "Год рождения", "Год смерти".

В этом случае ни одно из полей не является обязательным. Заполнение каждого из полей увеличивает количество ограничивающих условий для поиска. Допускается использование символов обобщения. Символ "%" означает любое количество символов на его месте, знак "_"означает ровно один символ. После окончания ввода следует нажать кнопку Submit ("Готово").

Если для данного критерия найдено несколько соответствий, то результаты поиска отображаются группами по пять штук, при этом внизу списка появляется кнопка Next ("Далее") для продолжения поиска.

Этот список содержит краткие сведения о каждой личности. Для получения более подробной информации и возможности ее редактирования, необходимо перейти по гиперссылке details ("Подробно"), находящейся напротив каждой найденной личности. В результате будет выведена вся имеющаяся информация о данной личности и элементы форм для возможности изменения.

Окно содержит три формы – одна форма для изменения списка фамилий личности, вторая для изменения списка сокращений, третья для изменения дополнительной информации.

Форма изменения фамилий устроена следующим образом. Оно состоит из окна ввода данных, поля "Новая фамилия", списка уже внесенных фамилий, включая написание и признак того, является ли это написание принятым. Имеется переключатель для выбора полей и кнопка подтверждения введенной информации. Если надо добавить

новую фамилию в список, следует выставить флажок переключателя рядом с полем "Новая фамилия", внести написание новой фамилии в поле ввода, установить соответствующее значение для флажка, обозначающего признак принятости фамилии. Подтвердить данные следует, нажав кнопку Apply ("Принять"). После изменения данных в базе окно будет обновлено.

Если необходимо изменить или удалить какую-либо фамилию, то надо поставить флажок переключателя напротив нее. Встроенный в код страницы обработчик событий переключателя произведет быстрый автоматический перенос данных о выбранной фамилии в поля ввода. Это значительно облегчает процедуру редактирования. После этого можно изменить написание или признак принятости, исправив соответствующие поля формы. Если убрать флажок в поле Keep ("Хранить"), фамилия будет удалена. Поэтому необходимо аккуратно следить за состоянием этого флажка. Подтверждение осуществляется нажатием кнопки Apply ("Принять").

Изменение сокращений осуществляется аналогичным образом.

В третьей форме содержатся три элемента для изменения дополнительной информации, года рождения и года смерти. После изменения какого-то из трех элементов или их комбинации для подтверждения изменений следует нажать кнопку Apply ("Принять").

Ввод данных о новой личности осуществляется через ту же форму, что и поиск. Если в раскрывающемся списке выбран пункт “добавить личность”, то в поля ввода необходимо внести соответствующую информацию. При этом поля “фамилия” и “сокращение” являются обязательными.

После занесения новой личности в базу будет произведено чтение информации из базы данных и в окне будет отображаться полная информация о только что введенной личности. При этом всю эту информацию можно непосредственно с этого момента редактировать.

Таким образом, редакторская работа с базой данных в диалоговом режиме осуществляется как последовательность внесений новых личностей или поисков внесенных ранее с последующим изменением некоторых атрибутов.

Работа пользователя

В этом режиме может производиться только получение информации, но не ее изменение. Поиск производится образом подобным поиску при работе в режиме редактора.

Из всех действий, допустимых в режиме редактора, в этом режиме оставлено только одно – простой поиск. Поэтому здесь

выбирать действие просто нет надобности, и такой выбор исключен. Остается только задавать критерии для поиска в соответствующих элементах форм. Аналогично предыдущему режиму производится разбиение на группы по пять личностей в каждой, если найдено более пяти человек. При этом в начале также выводится лишь краткая информация, и лишь затем по желанию пользователя может быть получена вся информация о некоторой личности, если этот пользователь запросит эту информацию через соответствующую гиперссылку (details). Естественно, отображение подробной информации полностью лишено элементов для редактирования.

Работа администратора

К функциям администратора относится заведение новых и изменение статуса "старых" пользователей, а также создание резервных копий БД, восстановление БД после возможных сбоев. Эту работу он может выполнять как через графический интерфейс системы, так и с помощью специальной утилиты.

Заключение

Представленная ИПС в ее нынешнем виде предназначена в основном для ботаников – специалистов по растениям семейства Зонтичных, но может быть легко адаптирована и для более широкого применения, а также других аналогичных задач. Кроме того, система используется в учебных целях для практикума по языку SQL на вечернем отделении факультета ВМиК В настоящее время база данных содержит сведения о написании фамилий и сокращений более 600 авторов таксонов растений семейства Зонтичных. Предполагается расширить систему набором утилит, позволяющих, в частности, получать сводки из базы в виде текстовых файлов в удобном для пользователей виде, в том числе генерировать справочник по исследователям растений семейства Зонтичных. После ввода в базу сокращений фамилий, использовавшихся во «Флоре СССР» и других авторитетных источниках будет организован авторизованный доступ к базе данных со страниц совместного сайта лаборатории вычислительного практикума и информационных систем факультета ВМиК и Отдела систематики и географии растений Ботанического сада МГУ (http://www.botanik.cs.msu.su/)

Литература

1. Леонов М.В. Комплекс программ для автоматизации исследований в таксономической ботанике. // Программные системы и инструменты. Тематический сборник № 2, М.: Изд-во факультета ВМиК МГУ, 2001, стр. 168-172.

2. Информационно-Поисковые системы в Зоологии и Ботанике. - Труды Зоологического Института РАН, Санкт-Петербург, 1999, Том 278, 139 стр.

Абрамов В.Г., Гранчак Т.В.

Автоматизация процесса составления расписания занятий в ВУЗе с помощью системы «Расписание»

Обзор подходов к автоматизации процесса составления расписания

Во многих областях деятельности человек сталкивается с задачей составления расписания, поэтому изучение проблемы началось сравнительно давно. Данная задача относится к классу NP-полных [1], т.е. для нее не известен полиномиальный алгоритм решения.

Существует несколько подходов к автоматизации процесса составления расписания. Наиболее радикальный подход – это полная автоматическая генерация расписания. Но в силу того, что задача нахождения оптимального расписания относится к классу NP–полных, для каждой реальной задачи найти точное решение за приемлемое время не удается. Все существующие программные системы предназначены для нахождения «достаточно хороших» решений. Мы называем решение «достаточно хорошим», если нас удовлетворяет его качество (критерии качества специфичны для каждой конкретной задачи). Для генерации расписаний используются различные алгоритмы: эвристические, генетические, поиск в глубину с возвратом, характерный для логического программирования, и др.

В качестве примеров можно привести систему HOREX [2], которая используется для создания расписаний в университете Монреаля, и систему EXAMINE [3], которая используется в нескольких университетах Канады. Обе эти системы строят расписание экзаменов с помощью эвристических алгоритмов, которые первоначально были разработаны для решения проблемы раскрашивания графа.

Более практичным представляется подход, при котором пользователь может управлять процессом составления расписания. В этом случае программная система используется в основном для редактирования расписания, а также помогает принять подходящее решение. К данному классу можно отнести программную систему Optime [0] для составления расписания экзаменов. Ее можно использовать в качестве редактора, кроме того, система может предложить пользователю различные варианты автоматически созданного расписания. Причем, автоматическую генерацию можно начать на любом этапе составления расписания.

Система «Расписание», используемая для составления расписания занятий на факультете Государственного управления МГУ, относится к классу специализированных редакторов. В основе системы лежит математическая модель, в соответствие с которой контролируются все вводимые пользователем (диспетчером) данные. Важно, что на каждом шаге составления расписания проверяется как корректность очередного шага, так и целостность и корректность всего расписания (см. раздел 0). Система не дает пользователю внести изменения, в результате которых расписание станет некорректным.

Модель расписания

В основе системы «Расписание» лежит математическая модель, которая описывает основные свойства ресурсов и правила их использования.

Ресурсы учебного расписания представлены в виде множеств: множество рабочих часов, множество преподавателей, множество студентов, множество аудиторий, множество предметов. В зависимости от конкретной задачи элементы каждого из этих множеств определяются по-разному. Например, в зависимости от учебного заведения, элементы множества рабочих часов удобнее представить как (Учебная пара, День недели) или как (Учебная пара, День недели, Четность недели). Свойства всех множеств описаны набором функций.

Само расписание представлено в виде функции занятия, аргументами которой являются ресурсы расписания.

Корректность расписания определяется с помощью системы ограничений. Ограничения представлены как соотношения между функциями, описывающими расписание и его ресурсы. Выделены две группы ограничений: жесткие и мягкие. Жесткие ограничения должны быть выполнены в расписании обязательно. Жесткие ограничения – это правила, определяющие корректное расписание. К мягким ограничениям мы относим те ограничения, которые желательно, но не обязательно соблюдать при составлении расписания. Степень выполнения мягких ограничений определяет качество расписания. Назовем расписание абсолютно корректным, если выполнены все мягкие и жесткие ограничения. Если мягкие ограничения выполнены лишь частично, то расписание - условно корректное.

Заложенные в программной системе «Расписание» алгоритмы проверки корректности определены в соответствии с математическим описанием ограничений в модели. Свойства ресурсов нашли свое отражение в структуре базы данных.

Приведем краткое описание модели.

Основные множества и их свойства. Множество рабочих часов (T)

Пусть "<" - отношение строгого порядка на T. Введем функцию предшествования *pred()*:

$pred: T \rightarrow T$,

$\forall t_1,t_2 \in T \;\; t_1=pred(t_2) \Leftrightarrow t_1<t_2$ и не $\exists t_3: t_1<t_3$ и $t_3<t_2$.

Введем функцию *Slots* - число учебных пар в определенном временном диапазоне:

$Slots: T \times T \rightarrow Z_0^+$ (Z_0^+ - множество неотрицательных целых чисел),

$\forall t_1,t_2 \in T$, таких что $t_1 \le t_2$: $Slots(t_1,t_2)=h \in Z^+$, причем $Slots(t_1,t_1)=1$,

$\forall t_1,t_2 \in T$, таких что $t_1 > t_2$: $Slots(t_1,t_2)=0$.

Пусть *Days* –множество дней недели. Введем функцию *Day*:

$Day: T \rightarrow Days$,

$\forall t \in T \;\; Day(t) \in Days$

Множество аудиторий (R)

Пусть Tr - множество типов аудиторий. Введем функцию *RoomType:*

$RoomType: R \rightarrow Tr$,

$\forall rm \in R \;\; RoomType(rm) \in Tr$

Введем функцию A_r:

$A_r: R \rightarrow T^*$ (T^* - множество подмножеств T),

$\forall rm \in R \;\; A_r(rm) \subseteq T$, допустимые часы, в которые данную аудиторию можно использовать.

Введем функцию *NeighbourRooms:*

$NeighbourRooms: R \times R \rightarrow \{0,1\}$,

$NeighbourRooms(rm_1,rm_2)=1$, если переход из аудитории rm_1 в аудиторию rm_2 требует значительного времени

$NeighbourRooms(rm_1,rm_2)=0$, если аудитории rm_1 и rm_2 находятся рядом.

Множество преподавателей (P)

Введем функцию A_p:

$A_p: P \rightarrow T^*$,

$\forall pr \in P \; A_p(pr) \subseteq T$, допустимые часы занятий преподавателя.

Введем функцию W_t:

W_t: $P \rightarrow T^*$,

$\forall pr \in P$ $W_t(pr) \subseteq T$, подмножество пар, которые удобны для проведения занятий. ($W_t(pr) \subseteq A_p(pr)$).

Множество предметов (D)

Введем функцию *SubjectType:*

SubjectType: $D \rightarrow Tr$,

$\forall d \in D$ $SubjectType(d) \in Tr$, каждому предмету соответствует определенный тип аудитории

Введем функцию *NeighbourSubjects:*

NeighbourSubjects: $D \times D \rightarrow \{0,1\}$,

$NeighbourSubjects(sj_1, sj_2)=1$, если предмет sj_2 нежелательно ставить сразу после sj_1 $NeighbourSubjects(sj_1, sj_2)=0$, если занятия по предметам sj_1, sj_2 можно ставить подряд

Множество студентов (S)

Пусть S^* - множество групп (это множество всевозможных подмножеств студентов).

$\forall gr_1, gr_2 \in S^*$ $gr_1 \cap gr_2 \neq \varnothing \Leftrightarrow \exists s \in S: s \in gr_1 \; u \; s \in gr_2$, группы могут пересекаться по студентам

Введем функцию $StudentProgram_t$, она задает рабочий план студента:

$StudentProgram_t : S \times D \rightarrow Z_0^+$,

$\forall s \in S$, $\forall d \in D$ $StudentProgram_t(s,d)=h \in Z_0^+$

Введем функцию $StudentProgram_{gr}$, она задает группу, в которой студент участвует в занятиях по каждому предмету:

$StudentProgram_{gr}$: $S \times D \rightarrow S^*$,

$\forall s \in S$, $\forall d \in D$ $StudentProgram_{gr}(s,d)=gr \in S^*$.

Если $StudentProgram_t(s,d)=0$, то $StudentProgram_{gr}(s,d)=\varnothing$.

Введем функцию *GroupProgram*, она задает число занятий в неделю, которые проводятся в каждой группе по каждому предмету:

GroupProgram: $S^* \times D \rightarrow Z_0^+$,

$\forall gr \in S^*$, $\forall d \in D$ $GroupProgram(gr,d)=h \in Z_0^+$.

$\forall s \in S$, $\forall d \in D$

$StudentProgram_t(s,d)=GroupProgram(StudentProgram_{gr}(s,d),d)$

Определим $G \subset S^*$, подмножество групп, которые назовем учебными группами:

$\forall gr \in S^*$ $gr \in G \Leftrightarrow \exists d \in D$: $GroupProgram(gr,d)>0$,

$\forall gr_1, gr_2 \in G \quad gr_1 \cap gr_2 \neq \varnothing \quad \Leftrightarrow \quad \exists s \in S, \quad \exists d_1, d_2 \in D$: $StudentProgram_{gr}(s, d_1) = gr_1$ и $StudentProgram_{gr}(s, d_2) = gr_2$

Определение занятия

Функция занятия *class* определена следующим образом:

$class: T \times D \times P \times G \times R \rightarrow \{0,1\}$, где

$class(t,d,pr,gr,rm)=1$ $\Leftrightarrow$ преподаватель *pr* ведет занятие в группе *gr* на паре *t* в аудитории *rm* по предмету *d*.

$class(t,d,pr,gr,rm)=0$ $\Leftrightarrow$ пятерка значений $(t,d,pr,gr,rm) \in T \times D \times P \times G \times R$ не соответствует никакому занятию.

Ограничения расписания

Жесткие ограничения

1. $\forall p \in P, \ \forall rm \in R, \ \forall gr \in G, \ \forall t \in T, \ \forall d \in D \ class(t,d,pr,gr,rm)=1$ только при условии, что $t \in A_p(pr) \cap A_r(rm)$

2. $\forall pr \in P, \ \forall t \in T$ $$\sum_{gr \in G, d \in D, rm \in R} class(t,d,pr,gr,rm) \leq 1$$

3. $\forall rm \in P, \ \forall t \in T$ $$\sum_{gr \in G, d \in D, pr \in P} class(t,d,pr,gr,rm) \leq 1$$

4. $\forall s \in S, \ \forall t \in T$ $$\sum_{\substack{gr: gr \in G \ u \ s \in gr, \\ d \in D, pr \in P, rm \in R}} class(t,d,pr,gr,rm) \leq 1$$

5. $\forall s \in S, \ \forall d \in D$ $$Student\ Pr\ ogram_{,}(s,d) = \sum_{pr \in P, t \in T, rm \in R} class(t,d,pr, Student\ Pr\ ogram_{gr}(s,d), rm)$$

Мягкие ограничения

1.$\forall$gr$\in$G, $\forall$pr$\in$P, $\forall$rm$\in$R, $\forall$t$\in$T, $\forall$d$\in$D class(t,d,pr,gr,rm)=1, при условии что SubjectType(d)= RoomType(rm).

2. $\forall$gr$\in$G, $\forall$pr$\in$P, $\forall$rm$\in$R, $\forall$t$\in$T, $\forall$d$\in$D class(t,d,pr,gr,rm)=1, при условии что $t \in W_t(pr)$.

3. $\forall s \in S$ и $\forall day \in Days$

$$\sum_{\substack{gr:gr \in G \ u \ s \in gr, \\ d \in D, pr \in P, rm \in R \\ Day(t)=day}} class(t,d,pr,gr,rm) = Slots(t_{\min}, t_{\max})$$

где

$$t_{\min} = \min_{\substack{t;Day(t)=day \ u \ class(t,d,pr,gr,rm)=1 \\ s \in gr, d \in D, pr \in P, rm \in R}} t \quad \text{и} \quad t_{\max} = \max_{\substack{t;Day(t)=day \ u \ class(t,d,pr,gr,rm)=1 \\ s \in gr, d \in D, pr \in P, rm \in R}} t$$

4. $\forall s \in S$, $\forall t_1, t_2 \in T$: $t_1 = pred(t_2)$, $Day(t_1) = Day(t_2)$, $\forall gr_1, gr_2 \in G$: $s \in gr_1$ u $s \in gr_2$, $\forall pr_1, pr_2 \in P$, $\forall d_1, d_2 \in D$

$class(t_1, d_1, pr_1, gr_1, rm_1) = 1$ и $class(t_2, d_2, pr_2, gr_2, rm_2) = 1$, при условии что $NeighbourSubjects(d_1, d_2) = 0$.

5. $\forall s \in S$, $\forall t_1, t_2 \in T$: $t_1 = pred(t_2)$, $Day(t_1) = Day(t_2)$, $\forall gr_1, gr_2 \in G$: $s \in gr_1$ u $s \in gr_2$, $\forall pr_1, pr_2 \in P$, $\forall d_1, d_2 \in D$

$class(t_1, d_1, pr_1, gr_1, rm_1) = 1$ и $class(t_2, d_2, pr_2, gr_2, rm_2) = 1$, при условии что $NeighbourRooms(rm_1, rm_2) = 0$.

Приведенная выше модель определяет основные элементы учебного расписания. Общее описание можно детализировать. От степени детализации модели зависит качество составляемого на ее основе расписания.

Подобная схема – представление ресурсов, ограничений, событий – может быть использована для описания расписаний и в других предметных областях.

Функциональные возможности системы «Расписание»

Система «Расписание», реализованная на основе модели, приведенной в разделе 0, уже год успешно функционирует на факультете Государственного управления МГУ. Ее использование значительно упрощает и ускоряет процесс составления расписания. Данная программная система берет на себя всю рутинную деятельность по проверке корректности и целостности расписания, что позволяет сосредоточить усилия диспетчера на учете трудно формализуемых требований.

Основной функцией системы «Расписание» является автоматизация процесса редактирования учебного расписания. Расписание составляется на основе информации о преподавателях, студентах, учебном плане, аудиториях. Система хранит все данные,

необходимые для создания расписания, а также предоставляет удобные средства для их редактирования. Таким образом, еще одной значительной функцией является управление информацией о ресурсах расписания. Кроме того, существует ряд дополнительных функций, которые помогают пользователю в процессе составления расписания (резервное копирование расписания, например). В следующих двух разделах работа с расписанием и управление информацией о его ресурсах описаны более подробно.

Работа с расписанием

Визуализация расписания

В расписании используются разные виды ресурсов, и на него можно смотреть с точки зрения каждого вида ресурсов. В программной системе «Расписание» выделены следующие представления расписания занятий:

- Расписание курса (с первого по пятый).
- Расписание аудиторий. Выделены два представления:
    - Расписание всех аудиторий
    - Расписание факультетских аудиторий
- Расписание преподавателя

Пользователь может редактировать расписание, представленное как расписание курса или как расписание аудиторий. Причем изменения, внесенные в одно из представлений расписания, будут отображаться в остальных. Расписания курсов и аудиторий отображаются в основном окне приложения. Расписание преподавателя будет выведено в отдельном окне по запросу пользователя, и используется для получения справочной информации. Расписание представлено в привычной для пользователя форме, т.е. в виде таблицы (сетки) занятий. Для расписаний курсов и аудиторий дни недели и учебные пары указаны в строках сетки. В столбцах сетки указаны номера групп (для расписаний курсов) или номера аудиторий (для расписания аудиторий) (рис.1).

Для расписания преподавателя дни недели указаны в столбцах сетки, а учебные пары – в строках.

Контроль над целостностью расписания

Одним из важнейших достоинств системы является контроль над целостностью расписания. При изменении данных (при добавлении, изменении занятий, изменении информации о преподавателях, об

аудиториях) выполняется проверка соблюдения следующих ограничений (в модели учебного расписания данные ограничения называются жесткими) (см. 0):

- Время проведения занятия должно быть допустимо для преподавателя
- Аудитория, используемая для занятия, должна быть доступна на время занятия
- Преподаватель может находиться только на одном занятии на каждой учебной паре
- Аудитория может использоваться только для одного занятия на каждой учебной паре
- Группа студентов может находиться только на одном занятии на каждой учебной паре

Выполнение этих условий является очень важным. Если они нарушены, то расписание не может быть использовано в реальной жизни.

Данная возможность системы упрощает работу диспетчера, составляющего расписание, т.к. пропадает необходимость проверять целостность расписания вручную. Причем, время, затрачиваемое на составление расписания, значительно сокращается.

Контроль над выполнением учебного плана

В системе «Расписание» каждому курсу студентов (с первого по пятый) поставлен в соответствие учебный план. Учебный план определяет число лекционных и семинарских часов, которые должны быть прочитаны в осеннем и весеннем семестре, по каждому предмету. Используя эту информацию, система осуществляет контроль над выполнением учебного плана.

На любом этапе составления расписания диспетчер может проверить, все ли занятия, необходимые по учебному плану, внесены в расписание, и нет ли «лишних часов» (имеется в виду ситуация, когда число занятий в расписании превышает число занятий, необходимых по учебному плану).

Эта возможность системы - еще одна ступень контроля корректности расписания. Чтобы проследить соответствие расписания учебному плану диспетчеру не нужно самостоятельно просматривать все расписание. Для получения статистики по любому предмету любого учебного курса достаточно выбрать эти курс и предмет в окне контроля над выполнением учебного плана.

Отчеты

На основе расписания, созданного с помощью данной программной системы, пользователь может получить ряд печатных отчетов, каждый из которых представляет некоторое подмножество множества занятий учебного расписания. С помощью математической модели учебного расписания множество занятий можно представить как $\{(t, d, pr, gr, rm) \mid (t, d, pr, gr, rm) \in T \times D \times P \times G \times R$ и $class(t,d,pr,gr,rm)=1\}$.

Приведем список отчетов и их математическое представление.

1. Расписание занятий курса (с первого по пятый)	$G' \subseteq G$ – группы одного курса, $\{(t, d, pr, gr, rm) \mid (t, d, pr, gr, rm) \in T \times D \times P \times G' \times R$ и $class(t,d,pr,gr,rm)=1\}$
2. Индивидуальное расписание каждого преподавателя	$p' \in P$ – преподаватель, $\{(t, d, pr, gr, rm) \mid (t, d, pr, gr, rm) \in T \times D \times \{p'\} \times G \times R$ и $class(t,d,pr,gr,rm)=1\}$
3. Расписание преподавателей одной кафедры	$P' \subseteq P$ – преподаватели одной кафедры, $\{(t, d, pr, gr, rm) \mid (t, d, pr, gr, rm) \in T \times D \times P' \times G \times R$ и $class(t,d,pr,gr,rm)=1\}$
4. Расписание всех аудиторий	$\{(t, d, pr, gr, rm) \mid (t, d, pr, gr, rm) \in T \times D \times P \times G \times R$ и $class(t,d,pr,gr,rm)=1\}$
5. Расписание факультетских аудиторий	$R' \subseteq R$ – факультетские аудитории, $\{(t, d, pr, gr, rm) \mid (t, d, pr, gr, rm) \in T \times D \times P \times G \times R'$ и $class(t,d,pr,gr,rm)=1\}$
6. Расписание аудиторий определенного типа	$tr \in Tr$ –тип аудитории, $\{(t, d, pr, gr, rm) \mid (t, d, pr, gr, rm) \in T \times D \times P \times G \times R$, $RoomType(rm)=tr$ и $class(t,d,pr,gr,rm)=1\}$

Управление информацией о ресурсах расписания. Обзор

Для составления расписания диспетчеру нужны сведения о:

- Преподавателях
- Учебном плане
- Группах студентов
- Аудиториях

Система «Расписание» хранит и предоставляет удобные средства для редактирования всей необходимой информации. Таким образом, приложение может использоваться в качестве справочной

системы для получения данных о преподавателях, учебном плане, группах студентов, аудиториях факультета.

Представление студентов в системе

Минимальная единица работы со студентами в системе «Расписание» - группа. Выделяются академические группы и неакадемические группы (например, группы по иностранным языкам, группы по информатике), в которые могут входить студенты разных академических групп. Академическая группа является исходным объектом для представления студентов. Такой выбор был обусловлен тем, что расписание обычно составляется относительно академических групп. Для каждой неакадемической группы пользователь может указать, студенты каких академических групп в нее входят.

Еще одним видом объединения студентов являются потоки. Каждый поток состоит из одной или нескольких академических групп. Причем, академические группы одного потока могут относиться к разным курсам. Это позволяет назначать общее занятие для студентов нескольких курсов.

Заметим, что при проверке целостности расписания система выдает предупреждение, если на одно и то же время назначены занятия нескольких групп и (или) потоков, куда входят студенты одной и той же академической группы.

Контроль использования аудиторного фонда

ФГУ имеет в своем распоряжении ограниченное число аудиторий (порядка десяти). Дополнительные аудитории, необходимые для проведения занятий, предоставляются другими факультетами МГУ. Таким образом, рациональное использование аудиторий имеет большое значение для управления факультетом.

Система «Расписание» предоставляет широкие возможности для работы с аудиториями. Все аудитории в системе делятся на факультетские и внешние. К факультетским относятся те аудитории, которые находятся в полном распоряжении факультета. К внешним относятся дополнительные аудитории, предоставляемые другими факультетами. Кроме того, существует деление аудиторий на типы (см. 0):

- Поточная
- Полупоточная
- Семинарская
- Языковая

- Компьютерный класс

Для каждой аудитории, как и для преподавателей, существует таблица занятости, с помощью которой администратор может задать время, в течение которого данную аудиторию можно использовать. Это особенно актуально в том случае, когда аудитория не находится в полном распоряжении факультета.

Для контроля использования аудиторий система предоставляет следующие возможности:

- Просмотр расписания аудиторий. Причем, пользователь имеет возможность просмотреть, как подробное, так и сжатое расписание аудиторий («Карту аудиторий»). Анализируя карту аудиторий, администратор сразу оценит ситуацию с аудиторным фондом: свободные, занятые и недоступные аудитории на каждой паре.
- Печать расписания аудиторий (см. раздел 0)
- Пользователь может получить информацию об аудиториях, которые свободны на определенной паре в определенный день недели
- Для аудиторий каждого типа система выдает отчет о количестве факультетских и внешних аудиторий, использующихся на каждой паре в каждый день недели

Хотя использование внешних аудиторий - скорее исключение, чем правило, в учебном процессе ВУЗа, ограниченные возможности аудиторного фонда являются одном из определяющих факторов при составлении расписания. Поэтому, в большинстве случаев, контроль использования аудиторий имеет важное значение.

Заключение

Реализация модели расписания может иметь свои особенности в программных системах, автоматизирующих составление расписания в разных учебных заведениях. Во-первых, существуют различные интерпретации множеств, описывающих ресурсы расписания; во-вторых, набор жестких ограничений, определяющих допустимое расписание, может меняться от системы к системе; в-третьих, набор формализованных и учитываемых системами мягких ограничений, определяющих качество расписания, также варьируется. Ограничения, являющиеся мягкими для расписания в одном учебном заведении, могут оказаться жесткими для расписания в другом учебном заведении. Степень реализации модели в системе определяет степень автоматизации процесса составления расписания.

Литература

1. Even S., Itai A., Shamir A. On the Complexity of Timetable and Multicommodaty Flow Problems // SIAM Journal on Computing. -1976.- 5, №4.

2. Desroches S., Laporte G., Rousseau J.M. HOREX: A Computer Program for the Construction of Examination Schedules // INFOR. – 1978.- 16, №.3.

3. Carter M.W., Laporte G., Chinneck. A General Examination Scheduling System // INTERFACES. – 1994.- 24, №.3.

http://www.asap.cs.nott.ac.uk/optime/

Расписание

Файл	Расписание	Редактрование	Данные	Сервис	Вид	Помощь

		101	102	103	104
ПОНЕДЕЛЬНИК	1	сем.Теория права Кашанина Т.В. ауд.419	сем.Защита и действия населения в ч/с		гр.G сем.Информатика Рязанов М.А. ауд.461
	2	лек.Теория права Кашанина Т.В. ауд.П-1			
	3	сем.Защита и действия населения в ч/с Филимонов Л.К.	сем.Введение в менеджмент Борисов В.К. ауд.455	сем.Введение в менеджмент Борисов В.К. ауд.419 (ч/н с 4.09)	сем.Теория права Кашанина Т.В. ауд.450
	4	гр.А сем.Информатика Абрамов В.Г. ауд.461	сем.Теория права Кашанина Т.В. ауд.450		сем.История мир. цивил. Макарова Е.И. ауд.455 (ч/н с 4.09)
				сем.Защита и действия населения в ч/с	сем.Эконом. и полит. география Мазуров Ю.Л. ауд.455 (ч/н с 11.09)
	5			гр.F сем.Информатика Абрамов В.Г. ауд.461	
ВТОРНИК	1	гр.2 сем.Англ.яз. Святова А.С. ауд.417	гр.3 сем.Англ.яз. Скибина Т.М. ауд.429	гр.5 сем.Англ.яз. Сыроватская Г.И. ауд.869	гр.7 сем.Англ.яз. Андреева С.И. ауд.495
		гр.1 сем.Англ.яз. Борисенко Т.И. ауд.419	гр.4 сем.Англ.яз. Скибина Т.М. ауд.429	гр.6 сем.Англ.яз. Погребенко Ю.И. ауд.463	гр.8 сем.Англ.яз. Данилина В.В. ауд.490
	2	гр.1 сем.Англ.яз. Борисенко Т.И. ауд.419	гр.3 сем.Англ.яз. Луканина М.В. ауд.450	гр.5 сем.Англ.яз. Сыроватская Г.И. ауд.869	гр.7 сем.Англ.яз. Андреева С.И. ауд.495

Рис.1

Полякова И.Н., Строилов Ю.В.

Система автоматического квазиреферирования

Введение

О проблеме реферирования

Закончившийся XX век часто называют веком информации. Это очень точное название – ведь именно в XX веке количество информации возросло в десятки раз. Это произошло еще до появления компьютеров, и для хранения и поиска информации людям приходилось держать огромные картотеки, хранилища с газетами, журналами и т.п.

Появление компьютеров не только чрезвычайно облегчило хранение, поиск и обработку информации, но и стимулировало дальнейшее увеличение количества документов. Перед многими людьми встала проблема: им регулярно приходится быстро просматривать большой объем документов и выбирать из них действительно нужные. Эта задача возникает и при работе с текстовыми базами данных, и при разборе электронной почты, и при поиске в Internet. Другая проблема - в крупных организациях, особенно государственных, правила делопроизводства предписывают сопровождать каждый документ кратким описанием.

В связи с этим возникла потребность в автоматическом составлении сжатых описаний документов – рефератов. Такие методы были разработаны, хотя, конечно, они не настолько совершенны, чтобы можно было получить полноценный реферат, в котором смысл исходного документа описан совсем другими словами. Сейчас состояние дел в этой области таково, что возможно автоматическое составление связных и максимально информативных квазирефератов заданного объема путем выделения из текста наиболее информативных предложений.

Постановка задачи

Целью данного проекта было создание системы, максимально точно и эффективно оценивающей значимость предложений в тексте. Система должна принимать документы в текстовом формате и на выходе давать, в зависимости от выбора пользователя, либо самые важные предложения, количество которых составляет заданный

процент от размера исходного документа, либо весь документ с расставленными оценками предложений. Должна быть возможность работы со словарем и создания пользовательских словарей.

Существующие методы автоматического реферирования

Проблема автоматического реферирования сравнительно новая и еще не имеет ни серьезных успехов, ни обширной истории. Авторы публикации [12], давая подробный обзор существующих технологий автоматического реферирования, приходят к выводу, что «в целом, отрасль средств реферирования находится в самом начале своего развития. Существует единое мнение о необходимости лучших методов оценки, однако, большинство задач еще не решено, в том числе, сохраняется необходимость в масштабируемых методологиях создания аннотаций».

В настоящее время одной из самых известных разработок в этой области является программа Libretto[8] компании МедиаЛингва. Эта программа может работать в двух режимах: составление рефератов русских и английских текстов и выделение из этих текстов ключевых слов. В режиме реферирования из текста отбираются предложения, в наибольшей степени характеризующие его содержание. В режиме выделения ключевых слов производится выборка из текста наиболее информативных слов.

Наиболее информативными словами Libretto считает слова, чаще других встречающиеся в тексте. При этом важность некоторого слова зависит и от его морфологических характеристик. Например, такие слова, как союзы, предлоги, междометия и т.п. не являются информативными и не учитываются при составлении реферата.

Еще одна используемая сейчас программа – TextAnalyst [9] - занимается не только реферированием. Она предлагает следующие возможности:

- построение семантической сети;
- построение тематической структуры текстов;
- реферирование текстов;
- гипертекстовая разметка текстов;
- смысловой поиск информации

Основная особенность этой программы – семантическая сеть, которая представляет собой множество понятий текста – слов и словосочетаний, связанных между собой по смыслу. В семантическую сеть включены не все слова текста, а лишь наиболее значимые, несущие

основную смысловую нагрузку. При этом в сеть не входят общеупотребительные слова, а также слова, очень редко встречавшиеся в тексте (частоту встречаемости может настраивать пользователь). Каждое понятие, многократно повторявшееся в различных местах текста, представляется в сети единственным элементом. Различные формы слов для отображения в один элемент сети приводятся к общей грамматической форме. К каждому понятию сети прилагается список других понятий, в сочетании с которыми оно встречалось в предложениях текста, а также список всех предложений, в которых употреблялось понятие.

В 2000 году произошло событие, которое не могли обойти вниманием специалисты, работающие в области квазиреферирования текста – был опубликован алгоритм работы программы Extractor[11], предназначенной для нахождения в тексте ключевых слов. Конечно, программы, занимающиеся квазиреферированием, работают немного по-другому, однако задача программы Extractor очень похожа, поэтому после незначительной адаптации есть возможность использовать многие идеи, примененные ее авторами. Одной из самых полезных идей оказалось выделение основы слова простым отбрасыванием последних нескольких букв.

Отметим, что, помимо традиционных на сегодняшний день методов составления квазирефератов, уже сейчас рассматривается также возможность анализа компьютером текста и составления его краткого изложения с использованием предложений, в текст не входящих. К сожалению, реализация таких методов достаточно затруднительна, а применение весьма требовательно к ресурсам. Но очевидно, что они представляют определенный интерес.

Первый из таких методов опирается на традиционный лингвистический метод разбора предложений. На первом шаге строится обычное дерево синтаксического разбора. После этого процедуры, которые занимаются реферированием, на основании семантической информации об аннотировании такого рода деревьев удаляют какие-то его части. В результате дерево существенно упрощается и становится, по сути, структурной выжимкой исходного предложения.

При использовании второго метода также проводится синтаксический разбор предложения, однако дерево разбора не порождается, а формируются структуры всей исходной информации, которые аккумулируются в текстовой базе знаний. В качестве структур могут быть использованы формулы логики предикатов или такие представления, как семантическая сеть или набор фреймов.

Там не менее, пока что основной задачей специалистов, работающих в области реферирования, остается разработка

оптимального алгоритма квазиреферирования. Хочется верить, что данная работа станет определенным шагом в этом направлении.

Алгоритм работы системы
Использованные в данном проекте методы

Среди всех опробованных программ самые качественные рефераты составляет программа Libretto. Однако при больших значениях коэффициента сжатия, когда из большого текста нужно выбрать одно-два предложения, результат ее работы редко оказывается удовлетворительным. В связи с этим было принято решение использовать в описываемой программе алгоритм, во многом аналогичный, но учитывающий недостатки и добавляющий новые возможности к алгоритму Libretto.

Так, в программе Libretto считается, что предложения, которые содержат анафорические слова «он», «этот» и т.п., являются зависимыми и, следовательно, менее значимыми. Такой подход не всегда представляется разумным. Как замечают авторы книги «Моделирование языковой деятельности в интеллектуальных системах» [3], в русском языке используется принцип экономии языковых средств. Одним из его следствий является употребление анафорических слов почти в каждом предложении текста. С другой стороны, главная мысль текста может выражаться и в зависимых предложениях. Поэтому было принято решение вообще отказаться от уменьшения их веса.

В качестве показателя значимости предложения Libretto рассматривает сумму показателей значимости слов, входящих в него. Но данный подход автоматически делает длинные предложения более значимыми. Однако часто бывает, что основная мысль текста выражается в коротком предложении, а в длинных идут пространные рассуждения, которые легко можно опустить. Поэтому в предлагаемой программе в качестве показателя значимости предложения используется средний показатель значимости слов. Значимость же слова вычисляется как количество употреблений этого слова в тексте, умноженное на коэффициент, различный для каждой части речи (По умолчанию наибольший коэффициент установлен для существительных и глаголов, нулевой для служебных слов. Однако пользователи могут изменять эти настройки).

Также предлагается учитывать, в каком месте текста находится предложение – чем ближе к началу оно находится, тем больше его вес. Имеет смысл увеличивать вес предложений, которые содержат ключевые фразы, такие как «в данной статье», «согласно результатам

анализа» и т.п. Некоторые слова, напротив, следует включать в стоп-лист и не учитывать их при подсчете весов предложений.

Алгоритм работы с текстом

В целом работа программы – после загрузки словаря, текста и задания параметров реферирования (формирование стоп-листа, задание коэффициента реферирования) – начинается с работы анализатора, разбивающего текст на предложения. Далее на вход непосредственно системе реферирования поступает последовательность лексем, разделенных специальными лексемами – знаками препинания, признаками конца предложения, конца абзаца и конца текста. Первой задачей программы является объединение поступающих лексем в предложения и формирование списка используемых слов. Лексемы, являющиеся знаками препинания, добавляются в предложение без изменений. Концевые признаки не добавляются, а являются сигналами, что нужно начинать новое предложение или заканчивать первичную обработку текста. Что же касается лексем, являющихся словами – для них необходимо выяснить и добавить к ним дополнительную информацию: часть речи и начальную форму слова. Для этого производится поиск слова в словаре (о работе со словарем речь пойдет ниже).

Если удается найти слово в словаре, проверяется, не входит ли оно в стоп-лист. Это нельзя было проверить раньше, потому что в стоп-лист вносятся слова в начальной форме, а мы не знаем начальную форму встретившегося в тексте слова, пока не найдем его в словаре. Если слово находится в стоп-листе или если оно не нашлось в словаре, оно описывается как служебное слово – таким образом,оно получит нулевой коэффициент значимости и не будет учитываться при подсчете веса предложения. Особо необходимо отметить такое поведение системы при отсутствии слова в словаре – эта возможность позволяет гибко настраивать словарь под конкретные задачи каждого пользователя. Так, если заранее известно, что текст, реферат которого нужно составить, частично посвящен компьютерной тематике и пользователя интересует именно эта часть текста, можно использовать словарь компьютерных слов, который позволит выбрать не просто самые значимые предложения, а именно те, в которых говорится об интересующей пользователя теме.

Итак, остается самый интересный случай – слово удалось найти в словаре и оно не входит в стоп-лист. В этом случае проверяется, не встречалось ли оно в тексте раньше. Если не встречалось – оно вносится

в список встречавшихся слов и ему присваивается вес 1. Если же встречалось, то есть его удалось найти в списке встречавшихся слов – его вес в этом списке увеличивается на единицу.

Таким образом, после первого прохода по тексту мы получаем, во-первых, список предложений и, во-вторых, список использованных слов из словаря с количеством их вхождений в текст. Теперь обходится список предложений и для каждого слова добавляется еще одна характеристика – его вес, то есть вес, указанный в списке использованных слов, умноженный на коэффициент значимости данной части речи. Затем подсчитывается средний вес слова в предложении и умножается на три коэффициента – близость предложения к началу текста, близость его к началу абзаца и наличие ключевой фразы. Полученное число и есть вес предложения.

После обхода списка предложений остается очевидная часть: берется количество предложений и умножается на коэффициент сжатия. Таким образом вычисляется количество предложений, которое должно войти в реферат – пусть их будет n. Затем формируется список этих предложений. Сначала в него включаются n первых предложений, затем продолжается просмотр списка предложений и, если встречается предложение, вес которого больше минимального веса в списке-результате, предложение с минимальным весом выбрасывается из этого списка, а в его конец добавляется найденное предложение. Список, который получится после завершения обхода списка предложений, и будет требуемым рефератом.

Алгоритм работы со словарем

Для работы со словарем предназначена специальная утилита. Функции добавления, удаления и исправления словарных слов доступны только из нее. Специальной возможностью этой утилиты, существенно облегчающей составление тематических словарей, является возможность просмотра текстов в обучающем режиме – программа проверяет, входит ли каждое слово из текста в активный словарь и, если не входит, предлагает его добавить. Таким образом можно получить словарь, максимально отвечающий потребностям пользователя, затратив минимум усилий.

При разработке представления словаря основной задачей было избежать получения полного списка словоформ для тех слов, которые заведомо не являются искомыми. То есть хотелось в большинстве случаев узнавать, может ли данное слово быть искомым, только по его начальной форме. Именно с этой целью используется упомянутый ранее метод программы Extractor. Размер части, которая отбрасывается от слова – три буквы. Исключение составляют слова, оканчивающиеся на

«ся» или «сь», то есть которые могут быть глаголами с возвратными частицами, для которых размер флексии может быть 5 букв – соответственно, именно столько необходимо отбросить, чтобы гарантированно осталась только основа. После можно просматривать только начальные формы слов – вернее, их левые части такой же длины, как и слово-образец с отброшенными последними буквами. Кроме того, достаточно просматривать только слова, длина начальной форма которых не больше чем на три буквы отличается от слова-образца (для возвратных глаголов также необходимым требованием является наличие у начальных форм возвратной частицы). И только при совпадении левых частей словарного слова и слова-образца и выполнении дополнительных ограничений возникает необходимость получить список словоформ словарного слова и проверить, не входит ли в него искомое слово – уже полностью, включая ранее отброшенные последние буквы.

Возможен вариант, что длина слова-образца была 3 буквы или меньше, то есть при отбрасывании последних букв от него ничего не остается. В этом случае нам не удается сузить область поиска за счет просмотра только слов с началом, совпадающим с началом образца. Однако второе сужение, по длине искомого словарного слова, остается в силе, то есть оказывается, что нам достаточно просматривать слова с длиной, не превышающей 6 букв – а это уже существенное сужение.

Еще одна интересная деталь словаря связана с обработкой исключений. Исключениями могут считаться как слова с чередованием букв в корне, так и, например, слово «он», словоформы которого никак не соответствуют начальной форме («его», «ему» и т.д.). Для того, чтобы поиск таких слов работал, был добавлен специальный тип слова – исключение. Вот как будет работать словарь со словом «он». Это слово заносится в словарь, как обычное местоимение, со всеми словоформами. Но все словоформы тоже заносятся в словарь, в качестве части речи для них ставится «исключение», а в качестве единственной словоформы – их начальная форма («он»). Поэтому функция поиска, найдя слово, тип которого «исключение», получает словоформы этого слова и вызывает сама себя, указывая в качестве аргумента первую (и единственную) словоформу. Полученное в результате этого вызова значение и будет являться начальной формой слова, содержащей всю информацию об этом слове, и именно оно будет возвращено в качестве результата поиска.

Некоторые детали реализации

Система реализована на Visual C++. Остановимся на некоторых моментах, которые призваны повысить эффективность работы системы.

Один из главных таких моментов – реализация и представление в памяти словаря. Очевидно, что он должен представляться в виде некоторой структуры, размер которой ограничен только объемом памяти компьютера. Поскольку первые версии системы писались под DOS, выбор был очевиден – списки. Однако хранение словаря в виде одного списка очень серьезно замедлило бы операцию поиска. Поэтому было решено пойти по аналогии с бумажными словарями и хранить по отдельности слова, начинающиеся на разные буквы. Таким образом, появился массив из 60 элементов типа `CPtrList`. 26 из этих списков предназначены для слов, начинающихся на латинские буквы, 33 – на русские буквы, и последний элемент – для всех остальных слов. Данная структура обладает некоторой избыточностью, так как вряд ли могут появиться слова, начинающиеся на «Ъ» или «Ь». Но вводить такое ограничение не хотелось, а хранение пустого списка не требует много памяти.

Первоначально предполагалось, что флексии, которые могут присоединяться к каждому слову, будут храниться в явном виде. Однако в процессе работы выяснилось, что данное решение очень неэкономно. Количество флексий в русском языке достаточно невелико, и разумнее было бы хранить их все в одном массиве, а в описании слова указывать индекс флексии в этом массиве. В связи с этим к описанию словаря добавился массив флексий (типа `CStringArray`). Для ссылки на его элементы используется тип `short`, поэтому количество его элементов ограничено 65535 – но вряд ли потребуется больше.

Итак, класс словаря описывается следующим образом (в редакторе словаря еще добавляются методы `addWord`, `removeWord` и `editWord`, назначение которых понятно из названий).

```
class CDictionary
{
public:
      CDictionary();
      virtual ~CDictionary();

// Attributes
public:
      CStringArray Endings;
      long count;
//    CDictWord* toPrint;
protected:
      CPtrList Words[60];

// Operations
public:
```

```
      bool findWord (CString word, int index,
CDictWord** result);
      CString getOrigForm(CDictWord& word);
      void getWordForms(CDictWord& word,
CStringArray& Forms);
      bool isWordForm(CString form, CDictWord&
word);
      void load();
};
```

Переменная `count`, не упоминавшаяся раньше, хранит количество слов в словаре и служит для вывода информации о загруженном словаре в статус-строке.

В словаре хранятся элементы класса `CDictWord` – словарные слова. Этот класс содержит переменные-члены для хранения основы, 32-х флексий и части речи. Флексии, как уже говорилось, представлены индексами массива `Endings` – то есть массив, хранящий их, описан как `short[32]`. Здесь экономнее использовать традиционный массив, а не один из классов `CArray`.

Еще один момент, которого хотелось бы коснуться – разбиение исходного текста на лексемы. Для этого сначала создается таблица символов, в которой каждому символу сопоставлено 16-битовое число, показывающее, является ли данный символ буквой, цифрой, знаком препинания и т.д. Ориентируясь на эти описания, анализатор сможет определять, какой тип присвоить создаваемой лексеме.

Выделением лексем занимается объект класса `CTokenizer` – в программе существует только один объект этого класса. Он может находиться в одном из состояний, описанных следующим типом:

```
enum _TokenizerState    {
      TZ_UNKNOWN,
      TZ_WORD,
      TZ_NUMBER,
      TZ_PUNCT,
      TZ_BLANK,
      TZ_PARAGRAPH,
      TZ_DELIM,
      TZ_OTHER
                  };

typedef enum _TokenizerState TokenizerState;
```

Класс содержит переменные `buf` (строковый буфер, содержащий считанные, но необработанные символы), `prev_char` (последний считанный символ, занесенный в буфер), `cur_char`

(символ, обрабатываемый в данный момент), next_char (символ, который будет обрабатываться следующим, но считываемый заранее, если дальнейшие действия обработчика зависят от него), State (текущее состояние обработчика).

Метод getTokenType позволяет определить, в зависимости от состояния обработчика, какой тип нужно присваивать очередной лексеме. Например, если текущее состояние – TZ_WORD, а в буфере нет ни одной буквы, лексеме присваивается тип TT_NONE, то есть «неизвестен». Если буфер длиннее одной буквы и состоит только из заглавных гласных букв, лексеме присваивается тип TT_ABBREV (который, как было указано при описании класса CToken, является подтипом класса TT_WORD, отличающийся от него только тем, что в старшем байте, определяющим расширенный тип, у него во втором бите единица, а у TT_WORD – ноль). Наконец, если ни одно из предыдущих условий не выполнено, лексеме присваивается тип TT_WORD.

Метод Stop останавливает обработчик, создает лексему, помещая в ее строковое значение текущее содержимое строкового буфера, а тип определяя с помощью метода getTokenType; очищает строковый буфер и переводит обработчик в новое состояние, которое является параметром метода.

Лексемы, которые создает обработчик, помещаются в буфер лексем tokenBuf и возвращаются вызывающей функции по запросу. Существует два метода, которые позволяют получить очередную лексему – peek, оставляющий возвращаемую лексему в буфере, и get, удаляющий ее. Оба этих метода сначала проверяют, пуст ли буфер лексем. Если он не пуст, возвращается первая лексема из буфера. В противном случае метод peek вызывает метод get, добавляет полученную лексему в буфер и возвращает ее. Таким образом, остается только рассмотреть метод get, вызванный в тот момент, когда буфер лексем пуст. В этом случае происходит считывание очередного символа из файла, его обработка в соответствии с его типом и текущим состоянием обработчика, создание новой лексемы и возвращение ее вызывающей функции.

Класс CSyntAnalizer, содержащий объект класса CTokenizer, получив от него лексему, определяет, является ли она последней в предложении. Если он считает, что да, он сохраняет этот результат и при обращении к нему за следующей лексемой генерирует и возвращает лексему специального вида, маркирующую конец предложения.

Рассмотрим работу системы со следующим фрагментом текста.

XX век часто называют веком информации. Это очень точное название - ведь именно в XX веке количество информации, которую необходимо обработать, возросло в десятки раз. Это произошло еще до появления компьютеров, и для хранения и поиска информации людям приходилось держать огромные картотеки, склады с газетами и т.п.

Появление персональных компьютеров не только чрезвычайно облегчило хранение, поиск и обработку информации, но и стимулировало дальнейшее увеличение ее количества. В связи с этим перед многими пользователями встала проблема: им регулярно приходится быстро просматривать большой объем документов и выбирать из них действительно нужные. Эта задача возникает и при работе с текстовыми базами данных, и при разборе электронной почты, и при поиске в Internet. Вторая проблема, которая возникает достаточно часто - это факт, что в крупных организациях, особенно государственных, правила делопроизводства предписывают сопровождать каждый документ кратким описанием.

В связи с этим возникла потребность в возможности автоматически составлять сжатые описания содержания документов - рефераты. Такие методы были разработаны, хотя, конечно, они не настолько совершенны, чтобы сформировать полноценный реферат, в котором смысл исходного документа может описываться совсем другими словами. Сейчас состояние дел в этой области таково, что возможно автоматическое составление связных и максимально информативных квазирефератов заданного объема путем выделения из текста наиболее информативных предложений.

Проблема автоматического реферирования сравнительно новая, и еще не имеет ни серьезных успехов, ни обширной истории. Авторы публикации «Системы автоматического реферирования» в журнале «Открытые системы» №12, 2000 г. (авторы - Удо Хан, адьюнкт-профессор университета Альберта Людвигса, Фрайбург (Германия), и Индерджиет Мани, научный директор подразделения Information Systems and Technology Division корпорации Mitre), давая подробный обзор существующих технологий автоматического реферирования, приходят к выводу, что «в целом, отрасль средств реферирования находится в самом начале своего развития. Существует единое мнение о необходимости лучших методов оценки, однако, большинство задач еще не решено, в том числе, сохраняется необходимость в масштабируемых методологиях создания аннотаций».

В настоящее время одной из самых известных разработок в этой области является программа Libretto компании МедиаЛингва. Эта программа может работать в двух режимах: составление рефератов

русских и английских текстов и выделение из этих текстов ключевых слов.

Ниже приведен этот же фрагмент после того, как система проставила веса слов и предложений. На обработку фрагмента на Pentium 166 ей потребовалось около секунды. Использовались следующие модификаторы: для слов – существительные и глаголы – 1.0, прилагательные, наречия и числительные – 0.5, местоимения – 0.25, служебные слова – 0.0; для предложений – 1.2 для первых трех абзацев и для предложений, содержащих ключевые слова. Вес слова стоит перед словом, к которому относится, заключен в угловые скобки и выделен курсивом. Вес предложения стоит перед предложением, заключен в квадратные скобки и выделен жирным.

Введение

[2.600000] *<1.00>* XX *<3.00>* век *<1.00>* часто *<1.00>* называют *<3.00>* веком *<4.00>* информации. **[1.283333]** *<2.50>* Это *<0.50>* очень *<0.50>* точное *<1.00>* название - *<0.00>* ведь *<0.50>* именно *<0.00>* в *<1.00>* XX *<3.00>* веке *<2.00>* количество *<4.00>* информации, *<0.75>* которую *<0.50>* необходимо *<1.00>* обработать, *<0.00>* возросло *<0.00>* в *<1.00>* десятки *<1.00>* раз. **[1.254545]** *<2.50>* Это *<1.00>* произошло *<0.00>* еще *<0.00>* до *<2.00>* появления *<2.00>* компьютеров, *<0.00>* и *<0.00>* для *<2.00>* хранения *<0.00>* и *<3.00>* поиска *<4.00>* информации *<0.00>* людям *<2.00>* приходилось *<1.00>* держать *<0.50>* огромные *<1.00>* картотеки, *<1.00>* склады *<0.00>* с *<1.00>* газетами *<0.00>* и *<0.00>* т.п.

[1.278571] *<2.00>* Появление *<0.50>* персональных *<2.00>* компьютеров *<0.00>* не *<0.50>* только *<0.50>* чрезвычайно *<1.00>* облегчило *<2.00>* хранение, *<3.00>* поиск *<0.00>* и *<1.00>* обработку *<4.00>* информации, *<0.00>* но *<0.00>* и *<1.00>* стимулировало *<0.50>* дальнейшее *<1.00>* увеличение *<0.75>* ее *<2.00>* количества. *<0.00>* В *<2.00>* связи *<0.00>* с *<2.50>* этим *<0.00>* перед *<0.00>* многими *<1.00>* пользователями *<1.00>* встала *<3.00>* проблема: *<0.75>* им *<0.50>* регулярно *<2.00>* приходится *<0.50>* быстро *<1.00>* просматривать *<0.50>* большой *<2.00>* объем *<4.00>* документов *<0.00>* и *<1.00>* выбирать *<0.00>* из *<0.25>* них *<0.50>* действительно *<0.50>* нужные. **[0.990000]** *<2.50>* Эта *<2.00>* задача *<2.00>* возникает *<0.00>* и *<0.00>* при *<1.00>* работе *<0.00>* с *<0.50>* текстовыми *<1.00>* базами *<1.00>* данных, *<0.00>* и *<0.00>* при *<1.00>* разборе *<0.50>* электронной *<1.00>* почты, *<0.00>* и *<0.00>* при *<3.00>* поиске *<0.00>* в *<1.00>* Internet. **[1.125000]** *<0.50>* Вторая *<3.00>*

проблема, <0.75> которая <2.00> возникает <0.50> достаточно <1.00> часто - <2.50> это <1.00> факт, <0.00> что <0.00> в <0.50> крупных <1.00> организациях, <0.50> особенно <0.50> государственных, <1.00> правила <1.00> делопроизводства <1.00> предписывают <1.00> сопровождать <0.50> каждый <4.00> документ <0.50> кратким <2.00> описанием.

[1.366667] <0.00> В <2.00> связи <0.00> с <2.50> этим <1.00> возникла <1.00> потребность <0.00> в <1.00> возможности <0.50> автоматически <1.00> составлять <0.50> сжатые <2.00> описания <2.00> содержания <4.00> документов - <3.00> рефераты. **[0.916667]** <0.50> Такие <2.00> методы <1.00> были <0.00> разработаны, <0.50> хотя, <0.50> конечно, <0.25> они <0.00> не <0.00> настолько <0.50> совершенны, <0.00> чтобы <1.00> сформировать <0.50> полноценный <3.00> реферат, <0.00> в <0.75> котором <1.00> смысл <0.50> исходного <4.00> документа <2.00> может <1.00> описываться <0.50> совсем <0.50> другими <2.00> словами. **[1.120000]** <0.50> Сейчас <1.00> состояние <1.00> дел <0.00> в <2.50> этой <2.00> области <0.50> таково, <0.00> что <0.50> возможно <2.00> автоматическое <2.00> составление <0.50> связных <0.00> и <0.50> максимально <1.00> информативных <1.00> квазирефератов <0.50> заданного <2.00> объема <1.00> путем <2.00> выделения <0.00> из <4.00> текста <0.50> наиболее <1.00> информативных <2.00> предложений.

[1.000000] <3.00> Проблема <2.00> автоматического <5.00> реферирования <0.50> сравнительно <0.50> новая, <0.00> и <0.00> еще <0.00> не <1.00> имеет <0.00> ни <0.50> серьезных <1.00> успехов, <0.00> ни <0.50> обширной <1.00> истории. **[0.902778]** <2.00> Авторы <1.00> публикации «<2.00> Системы <2.00> автоматического <5.00> реферирования» <0.00> в <1.00> журнале «<0.50> Открытые <2.00> системы» №12, 2000 <0.00> г. (<2.00> авторы - <0.00> Удо <0.00> Хан, <1.00> адьюнкт-профессор <1.00> университета <0.00> Альберта <0.00> Людвигса, <0.00> Фрайбург (<1.00> Германия), <0.00> и <0.00> Индерджиет <0.00> Мани, <0.50> научный <1.00> директор <1.00> подразделения <0.00> Information <0.00> Systems <0.00> and <0.00> Technology <0.00> Division <1.00> корпорации <0.00> Mitre), <1.00> давая <0.50> подробный <1.00> обзор <0.50> существующих <1.00> технологий <2.00> автоматического <5.00> реферирования, <1.00> приходят <0.00> к <1.00> выводу, <0.00> что «<0.00> в <0.50> целом, <1.00> отрасль <1.00> средств <5.00> реферирования <1.00> находится <0.00> в <1.00> самом <1.00> начале <0.25> своего <1.00> развития. **[0.802083]** <1.00>

Существует <0.50> единое <1.00> мнение <0.00> о <2.00> необходимости <0.50> лучших <2.00> методов <1.00> оценки, <0.50> однако, <1.00> большинство <2.00> задач <0.00> еще <0.00> не <0.00> решено, <0.00> в <0.25> том <1.00> числе, <1.00> сохраняется <2.00> необходимость <0.00> в <0.50> масштабируемых <1.00> методологиях <1.00> создания <1.00> аннотаций».

[0.937500] <0.00> В <0.50> настоящее <1.00> время <0.50> одной <0.00> из <1.00> самых <0.50> известных <1.00> разработок <0.00> в <2.50> этой <2.00> области <1.00> является <2.00> программа <1.00> Libretto <1.00> компании <1.00> МедиаЛингва. **[1.550000]** <2.50> Эта <2.00> программа <2.00> может <1.00> работать <0.00> в <0.50> двух <2.00> режимах: <2.00> составление <3.00> рефератов <0.50> русских <0.00> и <0.50> английских <4.00> текстов <0.00> и <2.00> выделение <0.00> из <2.50> этих <4.00> текстов <0.50> ключевых <2.00> слов.

При коэффициенте сжатия 0,33 реферат получается таким:

XX век часто называют веком информации. Это очень точное название - ведь именно в XX веке количество информации, которую необходимо обработать, возросло в десятки раз.

Появление персональных компьютеров не только чрезвычайно облегчило хранение, поиск и обработку информации, но и стимулировало дальнейшее увеличение ее количества.

В связи с этим возникла потребность в возможности автоматически составлять сжатые описания содержания документов - рефераты.

Эта программа может работать в двух режимах: составление рефератов русских и английских текстов и выделение из этих текстов ключевых слов.

Из этого примера видно, что системе удалось найти наиболее важные слова и предложения даже в таком коротком фрагменте. Выбранные предложения действительно отражают суть обработанного фрагмента. Конечно, реферат был бы понятнее, если бы удалось разрешить референцию «эта программа» и подставить вместо этого «программа Libretto», но проблема разрешения референций – отдельная большая проблема, задача решения которой не ставилась.

Понятно, что поскольку метод, использованный в системе – статистический, он будет работать тем лучше, чем больше текст, и если системе удалось справиться с коротким фрагментом, длинные тексты

(для которых она в первую очередь и предназначена), особенно после более точной настройки модификаторов, не являются для нее проблемой.

Необходимо отметить, что расширение возможностей системы осуществить достаточно легко. Так что при появлении компонентов, которые могут улучшить ее работу, добавить их в систему можно быстро и без особого труда.

Литература

1. Зализняк А.А. Грамматический словарь русского языка. М., Русский язык, 1977.
2. Мальковский М.Г, Грацианова Т.Ю., Полякова И.Н. «Прикладное программное обеспечение: системы автоматической обработки текстов». М., 2000.
3. «Моделирование языковой деятельности в интеллектуальных системах». М.: Наука, 1987
4. Прикладное языкознание. Учебник. СПб, Изд-во С-Петербургского университета, 1966
5. Нелюбин Л.Л. Компьютерная лингвистика и машинный перевод. М., ВЦП, 1991
6. Денисов П.Н. Лексика русского языка и принципы ее описания. М., Русский язык, 1993
7. Кривоносов А.Т. Язык, логика, мышление. М.-Нью-Йорк, Московский Лицей, 1996
8. Сайт компании «МедиаЛингва» – www.medialingua.ru
9. Сайт программы TextAnalyst – www.analyst.ru
10. Сайт research.metric.ru, посвященный технологиями анализа и поиска текстовой информации
11. Сайт программы Extractor – www.extractor.com
12. Хан У., Мани И. «Системы автоматического реферирования», статья в журнале «Открытые системы», №12, 2000 г.
13. Статья «Некоторые методы автоматического анализа естественного языка, используемые в промышленных продуктах» – www.citforum.ru/programming/digest/avtestlang.shtml

Аннотация

Брусенцов Н.П., Владимирова Ю.С. Троичное конструктное кодирование булевых выражений

Рассматривается компьютерное представление выражений булевой алгебры в совершенных нормальных формах ДК- и КД-шкалами на основе вектора битов и в произвольных нормальных формах аналогичными шкалами на основе векторов тритов.

Библиогр. 10 назв.

Доложено на Ломоносовских чтениях на факультете ВМиК МГУ 22 апреля 2002 г.

Брусенцов Н.П. Упорядочение булевой алгебры

Выявлением наряду с общепринятой монарной операцией отрицания-дополнения наиболее элементарной операции диаметральной инверсии терминов и выражений установлен неизбыточный базис булевой алгебры в составе операций (диаметральной) инверсии, конъюнкции и дизъюнкции. На этой основе осуществлены систематическое структурирование и интенсиональная интерпретация булевых выражений в логическом и теорико-множественном смысле.

Библиогр. 6 назв.

Доложено на Ломоносовских чтениях на факультете ВМиК МГУ 22 апреля 2002 г.

Леонов М.В., Мошкин К.Б., Леонов В.М XML как средство модернизации унаследованных баз данных. // Программные системы и инструменты. Тематический сборник № 3, М.: Изд-во факультета ВМиК МГУ, 2001.

В статье предложен подход к решению проблемы унаследованных баз данных и информационных систем с использованием языка XML. Представлена разработанная на основе такого подхода система, являющаяся модернизированным вариантом ИПС с базой данных формата dbf, разработанной около 15 лет назад для хранения таксономических данных по родам семейства Зонтичных.

Ил.:1. Библиогр.: 3 назв.

Петровский М.И. Мера сходства для сравнения прецедентов в системах анализа данных, поддерживающих стандарт OLEDB for DM

На настоящий момент разработано много алгоритмов классификации, кластеризации и выявления исключений, использующих различный математический аппарат. Большинство из

этих классических алгоритмов ориентированы на работу с данными простой структуры, когда объект классификации представляет собой вектор значений фиксированной длины. Но реальные задачи, решаемые с помощью систем анализа данных, приводят к более сложным структурированным данным. В этом случае объект классификации может представлять собой срез n-мерного куба или набор вложенных реляционных отношений. В стандарте OLEDB for DM предлагается формат для описания таких структур, но интерпретация этого описания ложится на алгоритм анализа данных. В связи с этим становится актуальной проблема определения меры сходства или расстояния на множестве таких объектов, это позволит использовать в системах анализа данных, поддерживающих стандарт OLEDB for DM, многие классические алгоритмы классификации, кластеризации и выявления исключений. В данной работе рассматривается подход к определению мера сходства с помощью метода потенциальных функций.

Машечкин И.В., Веселов Н.А. Оптимизация пропускной способности сетей, построенных по схеме Ч. Клоза, в условиях квазистатического размещения абонентов и отсутствия информации о загруженности системы

Данная статья посвящена исследованию методов управления соединениями в сетях передачи данных, построенных с использованием топологии Ч. Клоза. Для множества абонентов сети получено достаточное условие существования оптимального размещения виртуальных соединений, при котором данная схема не уступает полносвязной сети по всем характеристикам. Предложен эвристический метод построения оптимального размещения соединений абонентов, основанный на теории хроматических графов.

Ил.: 6. Библиогр.: 6 назв.

Доложено на Ломоносовских чтениях 2002 г. на факультете ВМиК МГУ.

Павлов Б.М., Новиков М.Д. Пакет программ для компьютерного моделирования пространственно-временных нелинейных процессов.

Описывается структура и возможности интерактивного пакета прикладных программ «Диссипативные структуры», ориентированного на проведение вычислительных экспериментов с математическими моделями типа «реакция-диффузия».

Ил.4. Библиогр.: 12 назв.

Гамаюнов Д. Ю. Современные некоммерческие средства обнаружения атак

В данной статье приводится обзор и сравнительный анализ нескольких существующих некоммерческих систем обнаружения атак (СОА) на компьютерные системы и сети. Проведен сравнительный анализ 7 систем по определенному набору критериев, который разрабатывался специально для сравнения и оценки качественных характеристик

Подшивалов А.Ю.Эффективные алгоритмы анализа взаимного расположения сильно развлетвленных пространственных объектов

Работа посвящена проблеме создания алгоритмов анализа взаимного расположения объектов, имеющих большие размеры и протяженность вдоль нескольких произвольно ориентированных осей (орбитальные станции и их сегменты). Спецификой алгоритмов является, с одной стороны, - необходимость учета ресурсов бортовых компьютеров, и, с другой стороны, - необходимость анализировать в реальном времени трехмерные сцены, содержащие сотни тысяч и миллионы примитивов. Кроме этого выдвигаются требования к высокой точности вычислений в связи с высоким риском столкновений объектов. В работе рассмотрены "жесткие" объекты (узлы сетки, описывающей поверхность, взаимно неподвижны), а также деформируемые объекты (включая объекты с подвижными "жесткими" конструкциями и объекты с произвольными деформациями). Рассматриваются различные виды деформаций, для каждого из которых представлены эффективные алгоритмы анализа столкновений.

•

Буряк Д.Ю., Визильтер Ю.В. Метод автоматизированного конструирования процедур обнаружения объектов на цифровых изображениях.

Данная статья посвящена проблеме автоматизированного конструирования процедур анализа изображений. В работе приведено описание созданного метода для автоматизированного построения близкой к оптимальной процедуры идентификации объекта на изображении по его эталонному изображению с использованием изображений обучающей выборки.

Ил.: 5. Библиогр.: 4 назв.

Доложено на конференции "Ломоносов-2002" на факультете ВМиК МГУ.

Зинченко Е.Ю., Попов А.М.. Исследование потери точности алгоритмов автоматической идентификации спикера по записи его речи. // Программные системы и инструменты. Тематический сборник № 1, М.: Изд-во факультета ВМиК МГУ, 2002.

В работе проводится исследование широко распространенного подхода к задаче идентификации спикера в условиях помех различной природы. Рассмотрены естественные помехи, ухудшающие качество микрофонной записи, а также активные помехи, связанные с присутствием других спикеров при записи.

Ил.: 8. Библиогр.: 9 назв.

Сальников А. Н. Сазонов А. Н. Карев М. В. Прототип системы разработки параллельных программ для гетерогенных многопроцессорных систем.

Предлагается инструмент поддержки параллельного программирования, который возьмёт на себя часть функций операционной системы по управлению выполнением параллельной программы на конкретной многопроцессорной системе с учётом информационной структуры задачи и архитектурных особенностей многопроцессорной системы. Для балансировки загрузки процессоров и загрузки каналов связи решается задача составления расписания работы многопроцессорной системы над графом алгоритма.

Библиография-15 названий

Сальников А. Н. Некоторые технические аспекты инструментальной системы для динамической балансировки загрузки процессоров и каналов связи.

На основе графа алгоритма строится параллельная программа, которая будет выполняться на многопроцессорной системе, причём узлы графа алгоритма выбираются для выполнения на процессоре динамически. Разрабатываемые программные утилиты призваны частично снизить потерю производительности при переносе параллельной программы на другую платформу. Предлагаемый инструмент поддержки параллельного программирования возьмёт на себя часть функций операционной системы и самостоятельно позаботится о выполнении параллельной программы на конкретной многопроцессорной системе с учётом информационной структуры задачи, самостоятельно решая задачу составления расписания работы многопроцессорной системы, а также задачу балансировки загрузки процессоров и загрузки каналов связи.

Библиография-16 названий

Вдовикин О.И., Машечкин И.В. О проблеме построения параллельной файловой системы для кластерных ВС.

Представлены подходы к организации параллельного ввода-вывода в высокопроизводительных ВС, рассмотрены вопросы использования их в кластерных установках. Изложен подход к

организации эффективной системы с реализацией необходимых параллельных интерфейсов с использованием прикладных библиотек, затронуты вопросы организации данных, программных компонент системы, обеспечения отказоустойчивости.

Библиография – 9 названий, иллюстраций – 7.

Доложено на Ломоносовских чтениях на факультете ВМиК МГУ 22 апреля 2002 г.

Леонов М.В., Глазов А.Э.ИПС по персоналиям ботаников с доступом через Интернет.

В статье представлены программные средства для автоматизации научных исследований в области таксономической ботаники, разработанные в лаборатории вычислительного практикума и информационных систем Факультета ВМиК МГУ для специалистов по растениям семейства Зонтичных Ботанического сада МГУ.

Библиогр.: 7 назв.

Абрамов В.Г., Гранчак Т.В. Автоматизация процесса составления расписания занятий в ВУЗе с помощью системы «Расписание» /. // Вестн. Моск. ун-та. Сер.15. Вычислительная математика и кибернетика.

Статья посвящена проблеме автоматизации процесса составления расписания на основе предлагаемой математической модели. В качестве реализации модели рассматривается система «Расписание», используемая на факультете Государственного управления МГУ

Полякова И.Н., Строилов Ю.В. Система автоматического квазиреферирования. //Программные системы и инструменты. Тематический сборник №3, М.: Изд-во факультета ВМиК МГУ, 2002.

В статье рассматривается система автоматического квазиреферирования, способная оценить значимость предложений в тексте и выделить наиболее важные из них. Дается обзор существующих систем, рассматриваются достоинства и недостатки их алгоритмов и предлагается новый алгоритм, учитывающий эти недостатки. Приводится пример работы системы, реализованной на основе предложенного алгоритма.

Библиогр.: 13 назв.

ДЛЯ ЗАМЕТОК

ДЛЯ ЗАМЕТОК

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 3

Под редакцией Л.Н. Королева

Издательский отдел
Факультета вычислительной математики и кибернетики
МГУ им. М.В. Ломоносова
Лицензия ИД N 05899 от 24.09.01 г.

119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова,
2-й учебный корпус

Напечатано с готового оригинал-макета
в издательстве ООО "МАКС Пресс"
Лицензия ИД N 00510 от 01.12.99 г.
Подписано к печати 02.12.2002 г.
Формат 60x88 1/16. Усл.печ.л. 14,0. Тираж 200 экз. Заказ 907.
Тел. 939-3890, 939-3891, 928-1042. Тел./Факс 939-3891.
119899, Москва, Воробьевы горы, МГУ.