МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ М.В. ЛОМОНОСОВА

Факультет вычислительной математики и кибернетики

**ТРУДЫ
ФАКУЛЬТЕТА ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ
И КИБЕРНЕТИКИ**

№ 4

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

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

**Москва
2003**

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: № 4.
/ Под ред. Л.Н. Королева. – М: Издательский
отдел факультета ВМиК МГУ (лицензия ИД №05899 от
24.09.2001г.), 2003. - 219с.

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 2003 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 5-89407-170-4

СОДЕРЖАНИЕ

СБОРНИК

" Программные системы и инструменты"

От редколлегии:

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 2003 на факультете ВМиК.

Перечень разделов сборника №4 таков:

- общие вопросы программирования и информатики
- машинная графика и обработка образов
- оптимизация вычислений
- интеллектуальный анализ данных
- инструментальные средства и сообщения

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

Редколлегия

Раздел I
Общие вопросы программирования и информатики

Ключников В.О., Костенко В.А., Маркин М.И.

Анализ эффективности нейросетевых, регрессионных и спектральных методов анализа временных рядов

1. Задача анализа временных рядов

Под временным рядом понимается последовательность наблюдений переменной x в моменты времени t_k, $k = \overline{1..n}$, n - число наблюдений.

Временной ряд $x_1, x_2, ..., x_n$ имеет следующие отличия от случайной выборки, образованной из наблюдений переменной x.

1. Последовательные значения $x_1, x_2, ..., x_n$ наблюдаются (обычно) через равные промежутки времени (тогда как в других методах не важна и часто не интересна привязка наблюдений ко времени).
2. Наблюдения $x_1, x_2, ..., x_n$, рассматриваемые как случайные величины, в большинстве случаев статистически зависимы.
3. Наблюдения временного ряда, в общем случае, не образуют стационарной последовательности, т.е. при изменении момента времени t_k (номера наблюдения k), изменяются числовые характеристики случайной переменной $x = x(t_k)$, в частности ее среднее значение и дисперсия: $\mu_x = \mu_x(t_k)$, $\sigma_x = \sigma_x(t_k)$ [1, 5].

Анализ временных рядов предполагает, что временной ряд содержит регулярную составляющую (обычно включающую несколько компонент) и нерегулярную (ошибку), которая затрудняет обнаружение регулярных составляющих [1, 4].

В рамках регулярной составляющей временного ряда обычно выделяют следующие компоненты:

- тренд - общая тенденция, движение на повышение или понижение,
- длиннопериодическая компонента - определяет отклонение от тренда с большим периодом,
- короткопериодическая компонента - определяет короткопериодические колебания относительно тренда и длиннопериодической компоненты [6, 9].

Нерегулярная компонента вызывает отклонения от хода отклика, определяемого трендовой, и периодическими составляющими. Она может быть рассмотрена как случайная, и потому непрогнозируемая; в терминах статистики ее можно считать ошибкой наблюдения и обрабатывать аналогично случайным ошибкам измерений в статистике.

Анализ временного ряда обычно подразумевает под собой решение следующих задач:

- построение модели, отражающей характерные особенности ряда с целью их изучения
- прогнозирование будущих значений ряда,
- выделение различных составляющих исходного ряда.

Методы анализа временных рядов (регрессионные, спектральные и нейросетевые) будем оценивать по следующим критериям:

- использование априорных сведений о процессе, породившем временной ряд,
- потенциально достижимая точность построенной модели ряда,
- возможность использования разнородной входной информации,
- вычислительная сложность,
- возможность выявления различных компонент ряда.

2. Предобработка исходных данных

На практике важным этапом анализа является предобработка данных, включающая в себя сглаживание исходного ряда (с целью более ясного визуального выделения тренда) и преобразования исходного ряда, позволяющие выделить скрытые периодичности (логарифмическое, экспоненциальное или, менее часто, полиномиальное преобразование данных) [3, 4, 5, 6].

Сглаживание всегда включает некоторый способ локального усреднения данных, при котором нерегулярные компоненты взаимно погашают друг друга. Самый общий метод сглаживания - *скользящее среднее,* в котором каждый член ряда заменяется простым или взвешенным средним k соседних членов. $MA_i(n) = \frac{1}{k}\sum_{t=(1-k)/2}^{t=(k-1)/2} x_{i+t}$, здесь $MA_i(n)$ – средневзвешенное значение i–го элемента ряда. Недостатком данного метода является потеря «крайних» значений ряда. Вместо среднего можно использовать медиану значений, попавших в окно. Основное преимущество медианного сглаживания, в сравнении со сглаживанием скользящим средним, состоит в том, что результаты становятся более устойчивыми к выбросам (имеющимся внутри окна).

Наряду со скользящим средним используется также метод экспоненциального сглаживания. В качестве взвешенных значений используются экспоненциально взвешенные скользящие средние по всему временному ряду, то есть сглаженное значение в любой точке ряда является некоторой функцией всех предшествующих наблюдаемых значений. $E_t = W \cdot x_i + (1 - W) \cdot E_{i-1}$, здесь E_t - экспоненциально сглаженное значение i -го члена ряда, W - коэффициент сглаживания ($0 < W < 1$). Коэффициент сглаживания значительно влияет на результаты, поэтому правильный выбор данного коэффициента является основной проблемой данного метода.

Предварительное функциональное преобразование исходных данных позволяет расширить область применения авторегрессионных моделей. Чаще всего применяют или логарифмирование $z_t = \ln x_t$, или переход к обратным величинам $z_t = \frac{1}{x_t}$ [1, 4].

3. Регрессионный анализ

Цель регрессионного анализа заключается в установлении зависимостей между несколькими показателями и описании этих связей достаточно простыми выражениями. Среди всех заданных показателей (признаков, переменных) один показатель считается результативным признаком y (откликом, выходом) и на этот показатель оказывают влияние остальные объясняющие переменные ξ_i. Данный метод статистического анализа основан на применении так называемого *метода наименьших квадратов* (МНК) [1, 3, 4, 5, 6].

В общем случае схему применения методов регрессионного анализа можно описать следующим образом:

- предполагается, что временной ряд порождается неизвестной функцией $y = f(\xi_1, \xi_2, ..., \xi_p)$ от факторов $\xi_1, \xi_2, ..., \xi_p$
- выбирается уравнение регрессии, например, для модели линейной регрессии $y = a_0 + a_1\xi_1 + ... + a_p\xi_p + \varepsilon$. Здесь $a_0, a_1, ..., a_p$ - параметры модели (регрессионные коэффициенты), ε - ошибка модели
- из условия минимума суммы квадратов невязок по всем обрабатываемым точкам ряда вычисляются параметры модели - $\Phi(a_0, a_1, ..., a_p) = \sum_i (y_i - x_i)^2 \longrightarrow \min$, где x_i - истинное значение i -го члена ряда, y_i - моделируемое значение.

- уравнение регрессии с найденными коэффициентами используется для прогнозирования временного ряда (вычисления значений функции $f(\xi_1,\xi_2,...,\xi_p)$ в соответствующих точках).

Основной проблемой регрессионного анализа является выбор "правильного" уравнения регрессии, от которой напрямую зависит точность и вычислительная сложность метода. Сложность, а часто, и возможность решения проблемы выбора "правильного" уравнения регрессии зависит от наличия каких-либо априорных данных об исходном ряде.

В общем случае методы регрессионного анализа временных рядов не предназначены для выделения различных компонент временного ряда.

4. Спектральный анализ

В спектральном анализе исследуются периодические модели данных. Цель анализа - разложить временные ряды с циклическими компонентами на несколько основных синусоидальных функций с определенной длиной волн, с целью выявления наиболее значимых компонент. В результате успешного анализа можно обнаружить всего несколько повторяющихся циклов различной длины в интересующих временных рядах, которые, на первый взгляд, выглядят как случайный шум [2, 6, 8].

Длина волны функций синуса или косинуса, как правило, выражается через частоту ν (число циклов в единицу времени). Период T функций синуса или косинуса определяется как продолжительность по времени полного цикла. Таким образом, это обратная величина к частоте - $T = \frac{1}{\nu}$.

Задача выявления тех частот, появление которых наиболее характерно для данного ряда может быть решена с помощью соответствующей задачи множественной линейной регрессии. В данной модели под независимыми переменными подразумеваются синусы и косинусы различных частот. Уравнения для данной модели множественной регрессии может быть записано следующим образом:

$y_t = a_0 + \sum_{k=1}^{q}\left[a_k \cdot \cos(\lambda_k \cdot t) + b_k \cdot \sin(\lambda_k \cdot t)\right]$, здесь λ_k - круговая частота, выраженная в радианах, $\lambda_k = 2 \cdot \pi \cdot \nu_k$, где $\lambda_k = 2 \cdot \pi \cdot \nu_k, \nu_k = \frac{k}{q}, q = \frac{n}{2}$, n – число наблюдений ряда [2, 8].

Фактически, вычислительная задача подгонки функций синусов и косинусов разных длин к данным может быть решена с помощью регрессионных методов. При этом, коэффициенты a_k при косинусах и b_k при синусах - это коэффициенты регрессии, показывающие степень, с которой соответствующие функции коррелируют с данными (сами синусы и косинусы на различных частотах не коррелированы или, ортогональны - таким образом, задача сводится к разложению по ортогональным полиномам).

В итоге, спектральный анализ определяет корреляцию функций синусов и косинусов различной частоты с наблюдаемыми данными. Если найденная корреляция (коэффициент при определенном синусе или косинусе) велика, то можно заключить, что существует строгая периодичность на соответствующей частоте в данных.

На практике, при анализе временных рядов строгая периодичность в данных часто отсутствует. В этом случае имеет смысл искать частоты с наибольшими спектральными плотностями, то есть частотные области, составленные из многих близких частот, которые вносят наиболее существенный вклад в поведение ряда. Точность воспроизведения исходного ряда можно регулировать путем удаления из ряда частот с малыми (по сравнению с остальными) амплитудами.

В случае, если повторяющихся циклов в данных нет, то есть, каждое наблюдение независимо от других наблюдений, то распределение частот будет близко к экспоненциальному. Этот эффект называется белым шумом.

Вообще говоря, спектральный анализ имеет смысл применять к стационарным рядам, то есть из исходного ряда полезно вычесть среднее и тренд. Иначе, спектральная плотность ряда «забьется» большим коэффициентом при нулевой частоте (коэффициентом a_0), ведь по сути, среднее – это цикл частоты 0 в единицу времени, то есть константа. Исходя из целей спектрального анализа (выделение циклических компонент в ряде), тренд также желательно удалить. Фактически, оба этих эффекта могут заслонить более интересные периодичности в данных. С другой стороны, методы спектрального анализа иногда применяются к исходным рядам, с целью выделения, например, тригонометрических трендов.

При использовании алгоритма быстрого преобразования Фурье (БПФ) для анализа ряда длиной n наблюдений время спектрального разложения пропорционально $n \cdot \log_k n$, где $k = \overline{2,4,8}$ - основание БПФ.

Построенная модель позволяет полностью воспроизвести исходный ряд по $\frac{n}{2}+1$ функциям косинуса и $\frac{n}{2}-1$ функциям синуса [2, 8].

Исходными данными для спектрального анализа служит сам ряд, таким образом, использование разнородной входной информации не представляется возможным.

5. Нейросетевые методы

Основной предпосылкой прогнозирования временных рядов с помощью нейронных сетей прямого распространения является известный факт из теории персептронов, который гласит о том, что многослойный персептрон с сигмоидной функцией активации нейронов может аппроксимировать *любую функцию?* со сколь угодно высокой точностью. Также, в теории персептронов доказывается, что для выполнения данной задачи в принципе достаточно всего лишь одного скрытого слоя нейронов с сигмоидной фунцией активации. Более того, такая сеть может аппроксимировать как саму функцию, так и ее производные. Точность аппроксимации возрастает с числом нейронов скрытого слоя. При N нейронах скрытого слоя ошибка аппроксимации оценивается как $O(\frac{1}{N})$ [7].

Процесс построения нейросетевой модели для прогнозирования временного ряда включает в себя следующие этапы:

- погружение в многомерное пространство,
- выделение эффективных признаков,
- обучение сети.

С помощью погружения исходного ряда в многомерное пространство задача прогнозирования временного ряда сводится к типичной задаче нейроанализа – аппроксимации функции многих переменных. Процедура погружения ряда в m - мерное пространство X^m состоит в следующем. Для k - го элемента ряда $x_1, x_2, ..., x_n$ соответствующий вектор $\mathbf{X}_k$ состоит из m значений ряда в последовательные моменты времени: $\mathbf{X}_k = (x_{k-1}, x_{k-2}, ..., x_{k-m})$. В теории динамических систем доказывается следующая теорема Такенса. Если временной ряд порождается динамической системой (то есть значение ряда $x_1, x_2, ..., x_n$ определяется функцией, зависящей от состояния системы), то существует такая глубина погружения d (вообще говоря, примерно равная числу эффективных степеней свободы данной системы), которая обеспечивает однозначное предсказание следующего значения временного ряда. Таким образом, выбрав достаточно большое d можно гарантировать однозначную зависимость значения k - го элемента ряда от d его предыдущих элементов, то есть $x_k = f(\mathbf{X}_k)$. В этих условиях задача

прогнозирование будущих значений временного ряда сводится к экстраполяции функции многих переменных. Нейронная сеть в этом случае используется для восстановления этой неизвестной функции по набору примеров, заданных историей данного ряда [7].

Этап выделения эффективных признаков необходим для того, чтобы уменьшить размерность входов нейронной сети. На данном этапе строится отображение входных векторов на агрегированные признаки, наиболее сжато описывающие входную информацию.

Обучение нейронной сети продолжается до тех пор, пока погрешность модели не будет достигать наперед заданного значения.

Прогнозы нейросетей, обученных с использованием локально-оптимальных алгоритмов обучения могут разниться из-за случайности начального выбора синаптических весов. Данный элемент неопределенности можно выгодно использовать, усредняя значения прогнозов по построенным моделям с различными начальными весами. Имея в виду неравенство Коши $(\frac{1}{N}\sum_{i=1}^{N}\varepsilon_i)^2 \le \frac{1}{N}\sum_{i=1}^{N}\varepsilon_i^{\,2}$, здесь ε_i - ошибка прогноза i - й сети, N - число построенных сетей, можно заключить, что ошибка усредненного прогноза будет не больше среднеквадратичой ошибки отдельных сетей. От средних значений можно перейти к взвешенным, где веса выбираются в зависимости от качества прогноза той или иной сети. Настройка весов нейронной сети производится исходя из максимизации качества взвешенного прогноза относительно исходной выборки. Используя данную методику можно добиться значительного повышения качества прогноза (по сравнению с отдельной сетью).

Преимущество нейросетевого моделирования заключается в том, что оно базируется исключительно на исходных данных, не привлекая при этом никаких априорных знаний о конкретном процессе. Эта черта нейросетей, однако, является и их основным недостатком, так как часто, обладая знаниями о процессе, исследователь может выгодно применить их при построении модели.

Также несомненным преимуществом является возможность использования разнородной входной информации (например, совокупность количественных и качественных характеристик изучаемого процесса). Процедура настройки весов нейросети в процессе обучения фактически эквивалента поиску параметров модели, наилучшим образом отражающей основные закономерности ряда. Таким образом, процесс выбора наилучшей модели автоматизирован, в отличие от регрессионного анализа, в котором модели во многом основаны на субъективных взглядах исследователя.

Аппарат нейронных сетей также может использоваться для выделения различных компонент исходного ряда. Для этих целей

используются, в основном, самообучающиеся и автоассоциативные сети [7].

6. Заключение

В таблице приведены обобщенные результаты возможностей использования нейросетевых, регрессионных и спектральных методов для решения задач анализа временных рядов.

Обобщая полученные выводы, можно заключить, что нейросетевые методы являются наиболее универсальными из рассмотренных. Возможность использования разнородной входной информации и разложения на главные компоненты выгодно отличает нейросетевые методы и существенно расширяет круг потенциально разрешимых задач анализа временных рядов.

	Регрессионный анализ	**Спектральный анализ**	**Нейросетевой анализ**
Необходимость использования априорных сведений	Да	Нет	Нет
Потенциально достижимая точность модели	Зависит от "правильного" выбора уравнения регрессии	Регулируется путем удаления частот с малыми амплитудами	Любая, наперед заданная
Использование разнородной входной информации	Возможно	Невозможно	Возможно
Вычислительная сложность	Зависит от модели	$n \cdot \log_k n$, где k - основание БПФ, n - число наблюдений	Зависит от модели, вычислительной сложности алгоритмов обучения
Выявление отдельных компонент ряда	Зависит от модели	Выявление периодических компонент	Разложение на главные компоненты

Литература

1. Дж. Бокс, Г. Дженкинс. Анализ временных рядов. Прогноз и управление. Вып. 1 и 2.- М.: Мир, 1974.

2. Г. Дженкинс, Д. Ваттс. Спектральный анализ и его приложения. Вып. 1 и 2. - М.: Наука, 1972.

3. Abraham, Ledolter. Statistical methods for forcasting. New York: Wiley, 1983.

4. М. Кендалл. Временные ряды. -М.: Финансы и статистика, 1981, -199 с.

5. М. Кендалл, А. Стюарт. Многомерный статистический анализ и временные ряды. -М.: Наука, 1976, -736с.

6. Anderson. Time series analysis and forecasting. London: Butterworths, 1976.

7. А. А. Ежов, С. А. Шумский. Нейрокомпьютинг и его применение в бизнесе. Москва, 1998.

8. Priestly. Spectral Analysis and Time Series. New York: Academic Press, 1981.

9. Кондратьев Н.Д. Проблемы экономической динамики.- М.: Экономика 1989.- 525 с.

Памяти сетевого программиста и журналиста

Сухомлина В.В.

Сухомлина А.И.

Метод построения корректной политики безопасности для систем электронного документооборота

В работе рассматривается математическая модель формальной семантики механизмов разграничения доступа в системах электронного документооборота. Для данной модели определены критерии корректности механизмов доступа и построенных на их основе политик безопасности, а также предложены алгоритмы проверки корректности механизмов и политик безопасности. Рассмотренный подход успешно апробирован на практике при создании систем документооборота производственного назначения.

Введение

Системы электронного документооборота (СЭД) относятся к классу наиболее ответственных корпоративных информационных систем, во многом определяющих уровень автоматизации и эффективность функционирования предприятий.

Важно отметить, что характерной чертой СЭД является интеграция в рамках одной системы по существу всей корпоративной информации и обеспечение регламентированного коллективного доступа к ней. Все это делает чрезвычайно актуальной разработку для СЭД высокоэффективных и надежных средств защиты информации, а также средств организации управляемого доступа к ней. Поэтому одной из важнейших компонент СЭД является подсистема управления информационной безопасностью, отвечающая, в частности, за реализацию разграничения доступа к информации с учетом полномочий различных классов пользователей. Такие функции, как правило, возлагаются на специальный модуль управления разграничением доступа.

Указанный модуль решает следующие основные задачи:

1. Вычисление текущих прав доступа субъекта (например, пользователя или представляющего его процесса) к информационным объектам при условии, что субъект может:

- входить в состав нескольких групп,
- иметь несколько ролей,

- принадлежать некоторому подразделению в оргструктуре организации.

2. Построение списка доступных для объекта субъектов.

3. Контроль прав доступа к журналам, каталогам, справочникам системы.

4. Взаимодействие с системой правил, задающих жизненный цикл документа с учетом того, что

- определенная роль может иметь доступ к объекту только в некоторых состояниях жизненного цикла документа,
- на права роли относительно доступа к объекту могут накладываться дополнительные ограничения, исходящие из структуры и семантики жизненного цикла документа.

5. Контекстную модификацию (если это предусмотрено) не полностью определенных запросов субъектов для формирования запросов, соответствующих политике безопасности с последующей передачей их на исполнение.

6. Обеспечение легкости использования средств администрирования подсистемы безопасности, включая динамическую модификацию интерфейса взаимодействия пользователя с системой в соответствие с динамикой изменения конфигурации и правил функционирования подсистемы безопасности.

7. Проверку корректности (полноты, непротиворечивости) средства и политик безопасности и пр.

Требование гарантированной корректности средств и политик безопасности приводит к необходимости использования математического аппарата при построении подобного рода механизмов.

Отправной точкой для ряда формальных моделей служат следующие аксиомы информационной безопасности [1, 2]:

Аксиома 1: Все вопросы безопасности информации описываются доступами субъектов к объектам.

Аксиома 2: Информационная система обработки данных называется безопасной, если она обеспечивает контроль доступа к информации так, что только надлежащим образом уполномоченные лица или процессы имеют право читать, писать, создавать или уничтожать информацию.

Для теоретического обоснования решений по разработке механизмов управляемого доступа к информационным и программным

ресурсам информационных систем разработано немало формальных моделей и подходов. Примерами известных работ в этой области являются работы [3-6].

Ниже описана математическая модель, предназначенная для описания семантики средств информационной безопасности информационных систем класса СЭД, а также рассмотрены основные алгоритмические решения, связанные с проверкой корректности создаваемых средств и политик безопасности.

1. Математическая модель

Описание математической модели начнем с введения используемых далее обозначений:

OBJ - множество информационных объектов или просто объектов;

SUB - множество субъектов, $SUB \subseteq OBJ$;

T - множество типов объектов;

Функция $\tau: OBJ \to T$ - каждому объекту ставит в соответствие его тип;

S - множество состояний объектов;

Функция $\psi: OBJ \to S$ - каждому объекту ставит в соответствие его текущее состояние;

OP - множество операций над объектами;

D = {TRUE=1, FALSE=0} – множество решений;

R - множество ролей;

Функция $\chi: SUB \times R \to D$, такая, что для $\forall u \in SUB$, $\forall r \in R$,

$$\chi(u, r) = \begin{cases} 1, & \text{субъект } u \text{ исполняет роль } r \\ 0, & \text{субъект } u \text{ не исполняет роль } r. \end{cases}$$

Для описания жизненного цикла объекта будем основываться на ряде понятий и определений из теории графов и автоматов.

Для каждого *типа объекта* определим единственный жизненный цикл. Тогда каждый объект в системе будет обладать единственным жизненным циклом, т.к. каждому объекту $o \in OBJ$ однозначно соответствует его тип $t = \tau(o)$, $t \in T$. Логика обработки объектов в системе задается совокупность всех соответствующих им жизненных циклов.

Зафиксируем тип объекта $t \in T$ и построим граф, соответствующий системе правил жизненного цикла относительно типа t.

Определение 1. Граф $G^{(t)} = (V^{(t)}, E^{(t)})$ - назовем ***графом жизненного цикла объекта*** (или ***графом переходов***), где

- множество вершин $V^{(t)} = \{v_1,\ldots, vm_t\}$ будем называть ***множеством допустимых состояний*** для объектов типа t, $V^{(t)} \subseteq S$, и
- множество ориентированных ребер $E^{(t)}= \{e_{ij} = (v_i, v_j) \mid v_i, v_j \in V^{(t)}$, дуга e_{ij} помечена операцией $op_{ij} \in OP\}$ - множеством допустимых переходов между состояниями.

Не ограничивая общности, можно считать, что среди элементов множества вершин в $V^{(t)}=\{v_1,\ldots, vm_t\}$, v_1 - начальное состояние, vm_t – конечное состояние.

Для исследования свойств графа, нам потребуется матрица достижимости (транзитивного замыкания) $B^{(t)}=\{b^{(t)}_{ij}\}$, которую определим следующим образом:

$b^{(t)}_{ij} = 1$, если существует ориентированный путь из v_i в v_j в графе $G^{(t)}$,

$b^{(t)}_{ij} = 0$, в противном случае.

Из построения матрицы достижимости $B^{(t)}$ вытекает следующее: $\forall t \in T$, $\#V^{(t)}=m_t$, $B^{(t)}$ имеет размерность $m_t \times m_t$ и $b^{(t)}_{ii} = 1$, $\forall i$, $1 \le i \le m_t$.

Теперь введем следующие определения.

Определение 2. Жизненный цикл типа объекта $t \in T$ ***корректен***, тогда и только тогда, когда для матрицы достижимости $B^{(t)}$, соответствующей графу перехода $G^{(t)}(V^{(t)}, E^{(t)})$, выполняются следующие условия:

$$b^{(t)}_{ij} = \begin{cases} 1, i=1, \forall j, 1 \le j \le m_t, & (1) \\ 1, \forall i, 2 \le i \le m_{t-1}, \exists j, 2 \le j \le m_t, \text{ такое что } i \ne j. & (2) \\ 1, \forall i, 1 \le i \le m_t, \forall j, 1 \le j \le m_t, i=j, & (3) \\ 0, \text{иначе}. \end{cases}$$

Таким образом, приведенное выше определение утверждает следующие свойства, которыми обладает корректный жизненный цикл типа объекта:

(1) – из v_1 достижимы все остальные вершины графа,

(2) – из любой вершины $v_i \in V^{(t)}/\{v_1, v_n\}$ достижима хотя бы одна $v_j \in V^{(t)}/\{v_1, v_i\}$,

(3) – из v_i достижима v_i, $\forall v_i \in V^{(t)}$ (по определению матрицы достижимости).

По существу данное определение эквивалентно следующим условиям:

1. найдется хотя бы одна суперпозиция преобразований (операций), переводящая объект из начального состояния в каждое из состояний $V^{(t)}$, кроме начального состояния;
2. если объект находится в некотором состоянии, и оно не конечное, то должно быть определено преобразование, переводящее объект в новое состояние.

Определение 3. Конфигурация системы – это множество графов переходов $\{G^{(t)} \mid \forall t \in T\}$.

Определение 4. Конфигурация системы корректна, если выполнены условия:

1. Для $\forall o \in OBJ$ определен только один жизненный цикл,
2. Для $\forall t \in T$ жизненный цикл относительно типа t корректен.

Так как все преобразования в системе инициируются субъектом, то большое значение для процесса маршрутизации объекта, осуществляемого в соответствии с его жизненным циклом, имеет учет прав доступа к данному объекту, которыми обладает субъект. Права доступа субъекта приписываются его ролям [см. информационную модель], и определяют возможности оперирования с объектами для субъекта, выполняющего конкретную роль.

Определение 5. ***Графом разрешения доступа*** относительно роли $r \in R$ и типа объекта $t \in T$ называется подграф $GR^{(tr)}=(VR^{(tr)}, ER^{(tr)})$ графа переходов $G^{(t)}$, построенный следующим образом:

- $VR^{(tr)}$ - множество состояний объекта, при которых он доступен для обработки субъектом (т.е. субъекту разрешена операция просмотра объектов типа t), выступающим в роли r, $VR^{(tr)} \subseteq V_t$,
- $ER^{(tr)}$ - множество переходов между состояниями, которые уполномочен производить субъект, исполняя приписанные переходам операции над объектом, $ER^{(tr)} \subseteq E^{(t)}$.

Введенное выше определение графа разрешения доступа служит средством моделирования метода разграничения доступа, основанного на разрешающих правилах.

На практике для назначения прав доступа администратором системы может оказаться также полезной модель запретов доступа, а не основанная на правилах доступа модель разрешений. Это может быть удобным, например, когда модель запретов содержит меньше правил, чем модель с правами доступа. При этом запреты для субъекта также

приписываются его ролям [см. информационную модель], и определяются по отношению к типу объекта.

Определение 6. ***Графом запрета доступа*** относительно роли $r \in R$ и типа объекта $t \in T$ называется подграф $GD^{(tr)} = (VD^{(tr)}, ED^{(tr)})$ графа переходов $G^{(t)}$, построенный следующим образом:

- $VD^{(tr)}$ - множество состояний объекта типа t, в которых он может быть в соответствии с жизненным циклом, $VD^{(tr)} = V^{(t)}$,
- $ED^{(tr)}$ - множество переходов между состояниями, которые не уполномочен производить исполнитель, $ED^{(tr)} \subseteq E^{(t)}$.

Таким образом, получаем, что для каждой роли $r \in R$ и каждого типа $t \in T$ в общем случае существуют два графа, один - $GR^{(tr)}$, определяющий модель разрешений доступа, другой - $GD^{(tr)}$ _ модель запрета доступа.

Как граф $GR^{(tr)}$, так и граф $GD^{(tr)}$ оба они являются подграфами графа переходов $G^{(t)}$, а следовательно возможны 4 случая:

В графе переходов $G^{(t)}$ зафиксируем дугу $e' \in E^{(t)}$:

1. $e' \in ER^{(tr)}$, $e' \in ED^{(tr)}$ – преобразование, соответствующее дуге e', одновременно разрешено и запрещено,
2. $e' \in ER^{(tr)}$, $e' \notin ED^{(tr)}$ – преобразование, соответствующее дуге e', разрешено,
3. $e' \notin ER^{(tr)}$, $e' \in ED^{(tr)}$ – преобразование, соответствующее дуге e', запрещено,
4. $e' \notin ER^{(tr)}$, $e' \notin ED^{(tr)}$ – преобразование, соответствующее дуге e', одновременно не разрешено и не запрещено.

Для принятия решения о доступе в случаях 1) и 4) будем использовать следующую функцию, применяемую, в частности, в случае разрешения конфликтных ситуаций:

$\lambda_т$: $R \rightarrow D$ – каждой роли соответствует доступ по умолчанию.

$\lambda_п$:$R \times T \rightarrow D$ – каждой паре <роль, тип объекта> соответствует доступ по умолчанию.

Функции $\lambda_т$, $\lambda_п$ задаются таблицами значений, причем при расширении множеств R и T, функции доопределяются на них (например, значением d = 0).

Конфликтная ситуация при принятии решения о праве доступа к объекту может возникнуть, например, когда область действия правил охватывает несколько операций или состояний или когда используются

при администрировании систем обе модели доступа (разрешительная и запретительная) одновременно. В этих случаях возможно возникновение неоднозначности, тогда для их разрешения и будем использовать функции $\lambda_т$ и $\lambda_п$.

В принципе возможно построение практических систем разграничения доступа, исключающих такого рода неоднозначности, но ценой этому может быть существенное ограничение средств для администрирования информационной безопасности или введение сложного и ресурсоемкого аппарата поддержки средств спецификации ограничений целостности системы.

Рассмотрим теперь применение построенного нами формального аппарата для вычисления прав роли $r \in R$ относительно типа объекта $t \in T$. Для этого введем еще одно определение.

Определение 7. ***Графом доступа*** относительно роли $r \in R$ и типа $t \in T$ называется подграф $GA^{(tr)} = (VA^{(tr)}, EA^{(tr)})$ графа переходов $G^{(t)}$, построенный следующим образом:

- $VA^{(tr)}$ - множество состояний объекта, при которых он доступен для обработки субъектом, выступающим в роли r, $VA^{(tr)} \subseteq V^{(t)}$ ($VA^{(tr)}$ состоит из вершин, смежных с дугами из $EA^{(tr)}$),
- $EA^{(tr)} \subseteq E^{(t)}$, $EA^{(tr)}$ - множество переходов между состояниями, которые уполномочен производить субъект, исполняющий роль r, при этом данное множество определим с помощью ниже заданной функции g(e') следующим образом:

$$EA^{(tr)} = \bigcup_{\substack{\forall e' \in E^{(t)}: \\ g(e')=1}} \{e'\}$$

где для $\forall e' \in E^{(t)}$ определим функцию g(e') таким образом:

$$g(e') = \begin{cases} 1, \text{ если } ((e' \in ER^{(tr)}) \,\&\, (e' \notin ED^{(tr)})) \vee \\ \quad ((e' \in ER^{(tr)}) \,\&\, (e' \in ED^{(tr)}) \,\&\, (\lambda_т(r) \vee \lambda_п(r, t))) \vee \\ \quad ((e' \notin ER^{(tr)}) \,\&\, (e' \notin ED^{(tr)}) \,\&\, (\lambda_т(r) \vee \lambda_п(r, t)))) \\ 0, \text{ иначе} \end{cases}$$

Тогда задача вычисления прав роли $r \in R$ относительно типа объекта $t \in T$ по существу сводится к нахождению ***графа доступа*** $GA^{(tr)}$ $(VA^{(tr)}, EA^{(tr)})$ относительно типа $t \in T$. Это позволяет далее использовать элементы теории графов и теории автоматов для анализа корректности механизмов разграничения доступа, построенных на основе рассмотренной модели, а также разработать хорошо обоснованные

алгоритмы реализации таких механизмов и определить оценку их алгоритмической сложности.

Определение 8. Конфигурация системы разграничения доступа – множество графов доступов $\{GA^{(tr)} \mid \forall t \in T, \forall r \in R\}$.

Определение 9. Конфигурация системы разграничения доступа ***корректна*** относительно типа объекта $t \in T$, если выполнены условия:

1. $\forall r \in R$, либо $\exists u \in SUB$: $\chi(u, r) = 1$, либо $GA^{(tr)} = \varnothing$;
2. жизненный цикл типа объекта t корректен;
3. $[\cup_{\forall r \in R} GA^{(tr)}] = G^{(t)}$

Поясним данное определение. По существу оно означает, что выполняются следующие условия:

1. для любой роли найдется субъект, который ее исполняет,
2. жизненный цикл типа объекта t корректен и граф переходов $G^{(t)}$ удовлетворяет всем свойствам определения 2,
3. для каждого из преобразований над объектами типа $t \in T$, найдется роль, которая имеет право на осуществление данного преобразования, или если объединить все права всех ролей в системе, то получим граф переходов для данного типа объектов $G^{(t)}$.

Определение 10. Конфигурация системы разграничения доступа ***корректна***, если выполнены условия:

1. $\forall r \in R$, либо $\exists u \in SUB$: $\chi(u, r) = 1$, либо $GA^{(tr)} = \varnothing$;
2. Конфигурация системы $\{G^{(t)} \mid \forall t \in T\}$ корректна;
3. $\forall t \in T, [\cup_{\forall r \in R} GA^{(tr)}] = G^{(t)}$

Определение 11. ***Графом доступа*** относительно субъекта $u \in SUB$ и типа $t \in T$ называется подграф $GA^{(tu)} = (VA^{(tu)}, EA^{(tu)})$ графа переходов $G^{(t)}$, построенный следующим образом:

$$\cup_{\forall r \in R:\ \chi(u, r) = 1} GA^{(tr)} = GA^{(tu)} \subseteq G^{(t)}$$

Отметим, что граф $GA^{(tu)}$ строится в момент выполнения запроса, а не хранится долговременно в системе. Однако применение метода кэширования к графам $GA^{(tu)}$ позволяет хранить некоторое время данные структуры в памяти системы, используя их для последующих однотипных запросов. Такая техника позволяет существенно оптимизировать реализационные аспекты механизма разграничения доступа. В частности, это относится к реализации групповых и

множественных запросов, так как для каждого элементарного запроса из группового и множественного запроса, строится только один граф доступа $GA^{(tu)}$, используемый на протяжении исполнения всего исходного запроса.

2. Динамика функционирования. Критерии корректности

Механизм запросов является основным средством реализации взаимодействия (интерфейса) пользователей с системой электронного документооборота и, таким образом, средством реализации динамики ее функционирования. Именно посредством запросов субъекты (некоторая подсистема или пользователь) запрашивают систему на выполнение операций над информационными объектами.

Все запросы в защищенной системе анализируются подсистемой безопасности, которая проверяет права субъекта – инициатора запроса, на возможность полного или частичного удовлетворения запроса, при необходимости модифицирует запрос в соответствии с правами пользователя и далее пересылает только разрешенную на выполнение часть запроса.

Определение 13. ***Элементарный запрос*** – запрос на преобразование информации вида $q = (u, o, op)$, где $u \in SUB$, $o \in OBJ$, $op \in OP$, после обработки которого система принимает решение $d \in D$, что можно описать с помощью отображения вида:

δ_a: SUBxOBJxOP→ D.

Определение 14. ***Групповой запрос (GroupQuery)*** – запрос на преобразование информации $q = (u, <o_1,\ldots,on>, op)$, где $u \in SUB$, $o_1,\ldots,on \in OBJ$, $op \in OP$, субъект u запрашивает одну и ту же операцию op над несколькими объектами $o_1,\ldots,on$, что можно описать с помощью отображения вида:

δ_g:SUBxOBJ^{n}xOP → D^n.

Групповой запрос рассматривается, как суперпозиция элементарных запросов, и подсистема безопасности принимает решение по каждому элементарному запросу в отдельности. Если надо чтобы данный групповой запрос прошел как единое целое (либо все элементарные запросы выполнились, либо нет), то это достигается внешними средствами, например, внутри некоторой транзакции посредством организации выполнения группового запроса с обеспечением отката транзакции, в случае, если хотя бы один элементарный запрос будет запрещен.

Определение 15. ***Множественный запрос (MultiQuery)*** – запрос на преобразование информации q = (u, o, <op_1,…,op_m>), где u∈SUB, o∈OBJ, op_1,…,op_m∈OP, субъект u запрашивает несколько операций op_1,…,op_m над одним объектом o, что можно описать с помощью отображения вида:

$$\delta_m: SUB \times OBJ \times OP^m \rightarrow D^m$$

Аналогично групповому, множественный запрос рассматривается, как суперпозиция элементарных, и подсистема безопасности принимает решение по каждому элементарному запросу в отдельности.

Критерии корректности модели

Далее перейдем к определению критериев корректности механизмов разграничения доступа, которые мы будем использовать для анализа этих механизмов на основе построенной нами модели.

Зафиксируем тип объекта t ∈T и построим конечный ***инициальный (разрешающий) автомат*** $M_t(OP,D,S',H,F,s_1)$, где

- На вход данного конечного автомата подается цепочка элементов из множества операций (OP - *входной алфавит*),
- D = {0, 1} - *выходной алфавит*,
- S′ ⊆ S - *множество состояний*,

Не ограничивая общности, можно считать, что среди элементов множества состояний S′ ={s_1,…, s_m}, s_1 ∈ S′ - *начальное состояние*, sm∈ S′ - *конечное*. Каждый элемент из множества S встречается не более одного раза в множестве S′.

- H - *функция переходов*, отображающая пары из декартова произведения OP×S′ в S′,
- F - *функцией выхода*, отображающая пары из OP× S′ в D.

Если над объектом в состоянии si, в соответствии с его жизненным циклом, операция осуществима op_j, то $H(s_i, op_j) = s_{ij}$, $F(s_i, op_j)=1$, иначе $H(s_i, op_j) = s_i$, $F(s_i, op_j)=0$.

Функции F, H – задаются таблицами вида:

H	*функция переходов*				
Состояния \ операции	op_1	…	op_j	…	op_m
s_1	s_{11}		s_{1j}		s_{1m}

…	…	…	…	…	…
Si	s_{i1}		s_{ii}		s_{im}
..	…	…	…	…	…
s_n					s_{nm}

F	*функция выхода*				
Состояния \ операции	op_1	…	op_i	…	op_m
s_1	d_{11}		d_1j		d_{1m}
…	…	…	…	…	…
si			d_{ii}		
..	…	…	…	…	…
s_n					d_{nm}

Таким образом, автомат может однозначно определять, являются ли входные последовательности запросов допустимыми или нет.

Данный автомат удобно задавать геометрически с помощью ориентированных графов GMt (диаграммой Мили, диаграмма Мура не подходит, т.к. каждое преобразование имеет результирующее значение, которое зависит как от состояния, так и от операции). При этом каждому состоянию из множества S′ сопоставляется вершина орграфа $V^{(Mt)}$, и каждой паре (s, op), где op∈OP, s∈S′ сопоставляется дуга $E^{(Mt)}$, идущая из вершины, соответствующей s, в вершину, соответствующую H(s, op). Этой дуге приписывается пометка (op,F(s, op)). Вершина, соответствующая начальному состоянию s_1, помечается (звездочкой).

Граф $G^{(Mt)}(V^{(Mt)}, E^{(Mt)})$ преобразуем к графу переходов $G^{(t)}(V^{(t)}, E^{(t)})$ следующим образом:

$$\varphi: G^{(Mt)}(V^{(Mt)}, E^{(Mt)}) \rightarrow G^{(t)}(V^{(t)}, E^{(t)})$$
$$V^{(t)} = V^{(Mt)},$$
$$E^{(t)} = \cup \{ e' \}$$
$$\forall e' = (s', s'') \in E^{(Mt)}, e' \text{ - помечено } op' \in OP:$$
$$F(s', op') = 1$$

Данное преобразование φ взаимнооднозначное и по заданному графу $G^{(t)}(V^{(t)},E^{(t)})$ восстанавливается граф $G^{(Mt)}(V^{(Mt)}, E^{(Mt)})$, точнее сказать, можно построить автомат $M^{(t)}$, а $G^{(Mt)}$ - его диаграмма Мили.

$$\varphi^{-1}: G^{(t)}(V^{(t)}, E^{(t)}) \rightarrow G^{(Mt)}(V^{(Mt)}, E^{(Mt)})$$

1. OP – входной алфавит, D = {0, 1},S′ = $V^{(t)}$,

2. $s^* = s_1$ – начальное состояние,
3. Значения функции F, H зададим следующим образом:

- для $\forall s' \in S'$, $\forall op' \in OP$, $F(s', op')=0$, $H(s', op')= s'$ (инициализация функций F и H),
- $\forall e'=(s', s'') \in E^{(t)}$. e' - помечено op', s', s'' $\in S'$, op' $\in OP$, $\Rightarrow F(s', op')=1$, $G(s', op')= s''$.

Критерий 1: Система разрешит субъекту $u \in SUB$ осуществить элементарный запрос q = (u, o, op), $o \in OBJ$, $op \in OP$, если в графе доступа $GA^{(tu)}$, $t=\tau(o)$, найдется дуга, соответствующая операции op, направленная от вершины графа $\psi(o)$, соответствующей текущему состоянию объекта o.

Доказательство:

Построим конечный распознаватель - автомат с конечным числом состояний, который отличает допустимые входные цепочки операций, поступающих от субъекта, от недопустимых.

Проанализируем поступивший запрос q = (u, o, op), $o \in OBJ$, $op \in OP$, $u \in SUB$.

По типу $t=\tau(o)$, $t \in T$ и субъекту $u \in SUB$ *найдем* граф доступа $GA^{(tu)}$ и *построим* автомат $M^{(tu)}(OP,D,S',F,G,s^*)$ такой что, если субъект может осуществить запрос q = (u, o, op), то на выходе элемент d=1, $d \in D$, причем, $op \in OP$ считаем входящим элементом.

1. Множество операций OP – входной алфавит,
2. D = {0, 1}– выходной алфавит,
3. $S' = V^{(tu)}$ – множество состояний,
4. $s^* = \psi(o)$ – начальное состояние,
5. значения функции F, H зададим следующим образом:

- для $\forall s_i \in S'$, $\forall op_i \in OP$, $F(s_i, op_i)=0$, $H(s_i, op_i)= s_i$ (инициализация функций F и H),
- $\forall e_k=(s_{l1}, s_{l2}) \in EA^{(tu)}$. e_k - помечено op_k, s_{l1}, $s_{l2} \in S'$, $op_k \in OP$, $\Rightarrow F(s_{l1}, op_k)=1$, $H(s_{l1}, op_k)= s_{l2}$,

граф $GM^{(tu)}$ – диаграмма Мили для автомата $M^{(tu)}$.

Таким образом, при поступлении запроса q, автомат на вход получает op, и можно вычислить функцию $F(s^*, op)= F(\psi(o), op)$, чтобы принять решение о разрешении доступа при $F(s^*, op')=1$ или о запрете при $F(s^*, op)=0$■

Критерий 1′: Система разрешит субъекту $u \in SUB$ осуществить множественный запрос $q=(u, o, <op_1,\ldots,op_m>)$, $o \in OBJ$, $op_1,\ldots,op_m \in OP$, если в графе доступа $GA^{(tu)}$, $t=\tau(o)$, найдется: дуга, соответствующая операции op_1, направленная от вершины $\psi(o)=s_1$ к вершине s_2,

дуга, соответствующая операции op_2, направленная от s_2 к вершине s_3, ...,

дуга, соответствующая операции op_i, направленная от s_i к вершине s_{i+1}, …,

дуга, соответствующая операции op_m, направленная от s_m,

причем $s_1 \le s_2 \le \ldots \le s_i \le \ldots \le s_m$.

Доказательство: *(аналогично критерию 1).*

Проанализируем поступивший запрос $q=(u, o, <op_1,\ldots,op_m>)$, $o \in OBJ$, $op_1,\ldots,op_m \in OP$, $u \in SUB$,

по типу $t=\tau(o)$, $t \in T$ и субъекту $u \in SUB$ *найдем* граф доступа $GA^{(tu)}$ и *аналогично (как в предыдущем доказательстве) построим* автомат $M^{(tu)}(OP,D,S',F,G,s^*)$, граф $GM^{(tu)}$ – диаграмма Мили для автомата $M^{(tu)}$.

Таким образом, при поступлении запроса q, автомат на вход получает последовательность операций $op_1,\ldots,op_m$, а его выходом является последовательность $d_1,\ldots,d_m$ решений, тогда алгоритм обработки запроса следующий:

k шаг ($1 \le k \le m$):

если $F(\psi(o),op_k)=1$, система принимает решение о разрешении осуществления элементарного запроса $q=(u, o, op_k)$, то переходим на шаг k+1, иначе система принимает решение о запрете осуществления запроса $q = (u, o, op_k)$ и переходит на шаг m+1;

m+1 шаг: конец.■

Замечание 1: Для принятия решение об осуществлении субъектом $u \in SUB$ группового запроса $q = (u, <o_1,\ldots,o_n>, op)$, $o_1,\ldots,on \in OBJ$, $op \in OP$, запрос q рассматривается, как последовательность элементарных $q_1,\ldots,qn$, где $q_i = (u, o_i, op)$ и решение принимается по каждому запросу в отдельности.

3. Расширение модели с учетом динамики функционирования системы. Алгоритмы проверки корректности модели

В этом разделе рассмотрим те аспекты, касающиеся нашей модели, которые связаны с поддержанием ее в актуальном состоянии в процессе динамики функционирования, когда происходят изменения множеств и графов (дополнение или исключение элементов) модели.

Расширение/сужение множеств и изменение графов приводит к изменению системной конфигурации, что, в свою очередь, может потребовать полного или частичного перевычисления условий корректности модели, модификации других ее компонент, зависящих от выполняемых изменений.

Рассмотрим возможные изменения конфигурации модели и следствия, вызываемые этими изменениями.

- $OBJ' = OBJ \cup \{o^*\}$,
  $OBJ' = OBJ - \{o^*\}$, $o^* \in OBJ$,
  $SUB' = SUB \cup \{u^*\}$,
  $S' = S \cup \{s^*\}$,
  $OP' = OP \cup \{op^*\}$.

  Данные операции не влияют на результат вычисления выражений из определений 2, 4, 9, 10, а, следовательно, на корректность конфигурации системы разграничения доступа.

- $SUB' = SUB - \{u^*\}$, $u^* \in SUB$

  Проверка условия для множества SUB':
  $\forall r \in R$, либо $\exists u \in SUB'$: $\chi(u, r) = 1$, либо $GA^{(r)} = \varnothing$.

  Если условие выполнено, то можно сузить множество SUB:

  $SUB' = SUB - \{u^*\}$, $u^* \in SUB$.

- $S' = S - \{s^*\}$, $s^* \in S$

Если найдется хотя бы один граф среди графов переходов, разрешения или запрета доступа, в котором есть дуга, исходящая из вершины, соответствующей состоянию s^*, то множество состояний сузить относительно элемента s^* не представляется возможным, в противном случае возможно.

- $OP' = OP - \{op^*\}$, $op^* \in OP$

- Если найдется хотя бы один граф среди графов переходов, разрешения или запрета доступа, в котором есть дуга, помеченная операцией op*, то множество OP сузить относительно элемента op* не представляется возможным, в противном случае возможно.

- $T' = T \cup \{t^*\}$

1. Расширить множества типов: $T' = T \cup \{t^*\}$.
2. Построить граф переходов $G^{(t^*)}$ по определению 1.
3. Доопределить, если необходимо, до корректного графа (см. определение 2).
4. Можно переопределить $\lambda_\pi(r, t)$ значения функции в точках $\{(r, t) | \forall r \in R, t=t^*\}$.
5. Можно переопределить графы доступа, входящие в множество $\{GA^{(t^*r)} | \forall r \in R\}$ (из значений функций $\lambda_r(r)$ и $\lambda_\pi(r, t)$ вытекает, что графы доступа из множества $\{GA^{(t^*r)} | \forall r \in R\}$ либо пустые, либо совпадают $G^{(t^*)}$.

- $T' = T - \{t^*\}$, $t^* \in T$

1. Найти множество L_t объектов типа t*.
2. Удалить объекты, входящие в L_t из OBJ.
3. При сужении множества T: $T' = T - \{t^*\}$, $t^* \in T$, конфигурации системы $\{G^{(t)} | \forall t \in T\} - G^{(t^*)}$ и конфигурации системы разграничения доступа $\{GA^{(tr)} | \forall t \in T, \forall r \in R\} - \{GA^{(tr)} | t=t^*, \forall r \in R\}$ также сужается.

В данном случае, если множество объектов содержит хотя бы один объект типа t*, то множество T нельзя сузить относительно типа t*. С другой стороны, для каждого типа обязательно есть граф переход $G^{(t^*)}$, задающий жизненный цикл объектов типа t*, и семейство графов доступа $\{GA^{(tr)} | t=t^*, \forall r \in R\}$. Удаление всех этих графов идет в одной транзакции с удалением типа t*.

- $R' = R \cup \{r^*\}$

1. Расширить множества ролей: $R' = R \cup \{r^*\}$.
2. Найти множество субъектов, которые должны исполнять роль: $L_u \subseteq SUB$.
3. Переопределить функцию следующим образом:

$$\chi(u, r^*) = 1, \forall u \in L_u, r^* \in R;$$

4. Определить функцию на значении $r^* \in R$: $\lambda_r(r^*)=d$, $d \in D$.

5. Если $L_u \neq \varnothing$, то определить графы доступа, входящие в множество $\{GA^{(tr^*)} \mid \forall t \in T\}$ и дополнить ими конфигурации системы разграничения доступа.

6. Если $L_u = \varnothing$, то определить графы доступа, входящие в множество $\{GA^{(tr^*)} = \varnothing \mid \forall t \in T\}$ и дополнить ими конфигурации системы разграничения доступа.

7. Множество $\{GA^{(tr^*)} = \varnothing \mid \forall t \in T\}$, можно задать семейством функций: $\{\lambda_{Tr^*t}(r^*, t) = 0 \mid \forall t \in T\}$.

- $R' = R - \{r^*\}$, $r^* \in R$

1. Найти множество субъектов, исполняющих роль $r^* \in R$:
2. $L_u = \cup\{u\}$
3. $\forall u \in SUB$:
4. $\chi(u, r^*) = 1$

5. Переопределить функцию следующим образом: $\chi(u, r^*) = 0$, $\forall u \in L_u$.

6. Удалить все графы доступа, входящие в множество $\{GA^{(tr^*)} \mid \forall t \in T\}$ из конфигурации системы разграничения доступа.

7. $R' = R - \{r^*\}$, $r^* \in R$.

- Добавление или удаление дуг в графы, такие как граф переходов $G^{(t)}$, граф разрешения доступа $GR^{(tr)}$, граф запрета доступа $GD^{(tr)}$, граф доступа $GA^{(tr)}$, удовлетворяет соответствующим определениям.

Все эти изменения модели требуют повторного применения алгоритмов анализа корректности модели, т.е. проверки конфигурации системы разграничения доступа на корректность $\{ GA^{(tr)} \mid \forall t \in T, \forall r \in R\}$.

В системе фиксируется:

1. время последней модификации для каждого объекта в системе;
2. время фиксации последней успешной проверки корректности конфигурации системы разграничения доступа.

Каждый раз проверка всей системы на корректность является далеко не оптимальным решением. С целью оптимизации такого решения будем использовать механизм временных меток, связываемых со временем изменения системной конфигурации и фиксируемых с момента последней успешной проверки системы разграничения доступа на корректность, которая включает в себя проверку всех условий Определения 10. Данный механизм позволяет локализовать область

системной конфигурации, подлежащей проверке на корректность, и сократить тем самым объем вычислений по сравнению с тем, который был бы необходим для анализа полной конфигурации системы.

Следуя определению корректности системы, введенному в нашей модели, общая схема проверки логической правильности системной конфигурации может быть выражена следующим образом:

Пусть Ω = {OBJ, SUB, S, OP, T, R}

1. Найти множества $\Omega_k \subseteq \Omega$, которые были изменены после предыдущей проверки корректности.

2. Найти элементы множеств Ω_k, с пометкой времени последней модификации большей, чем время фиксации последней успешной проверки корректности конфигурации.

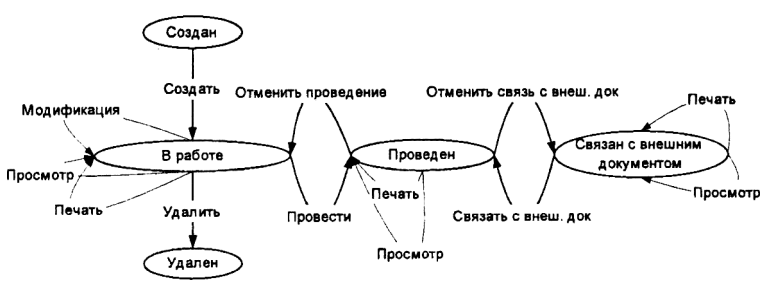
3. Применить к выбранным элементам множеств Ω_k алгоритмы проверки корректности описанные выше.

4. Примеры функционирования модели

Применение построенной модели проиллюстрируем на следующих примерах и задачах:

Пример 1:

Рассмотрим жизненный цикл типа объекта t:



Сопоставим состояниям вершины графа переходов $G^{(t)}$:

Состояние объекта	Вершина
Создан	v1
В работе	v2
Проведен	v3
Связан с внешним документом	v4
Удален	v5

Построим матрицу достижимости:

	v1	v2	v3	v4	v5
v1	1	1	1	1	1
v2		1	1	1	1
v3		1	1	1	1
v4		1	1	1	1
v5					1

По определению 2 жизненный цикл объектов типа t корректен.

Задача 1:

Системный запрос – задается функцией вида γ_a: RxOBJxOP→ D

Вход: объект o∈OBJ, операция op∈OP, роль r∈R

Выход: решение d∈D

Типу t=ft(o), t∈T и роли r∈R найдем граф доступа $GA^{(tr)}$

d = 1, если ∃ e′=(ψ(o), s′) ∈ $EA^{(tr)}$, дуга e′ помечена операцией op∈OP, s′∈ S′.

Задача 2: Определение для объекта o ∈OBJ список субъектов, которые могут выполнить хотя бы одну операцию над ним.

Вход: объект o ∈OBJ

Выход: Список субъектов Lu, , которые могут выполнить хотя бы одну операцию над объектом o ∈OBJ.

Решение:

Вариант 1:

Построим список субъектов L_u:

$$L_u = \cup\{u\}$$

$$\forall u \in SUB, \forall op \in OP:$$

$$\delta_a(u, o, op)=1$$

Вариант 2:

Найдем множество ролей L_r, которые могут выполнить хотя бы одну операцию над объектом o $\in$OBJ.

$$L_r = \cup\{r\}$$

$$\forall r \in R, \forall op \in OP:$$

$$\gamma a(r, o, op)=1$$

Построим список субъектов L_u, которые исполняют хотя бы одну роль из множества L_r.

$$L_u = \cup\{u\}$$

$$\forall u \in SUB, \forall r \in L_r:$$

$$\chi(u, r) =1$$

Задача 3: Определение прав доступа субъекта u$\in$SUB к журналу (или каталогу, или справочнику).

Вход: список объектов Lo=$\{o_1,\ldots o_m\}$, отобранных из множества объектов по некоторому критерию в журнал, каталог или справочник, и субъект u$\in$SUB

Выход: список операций Lop

Решение:

1. Найдем множество субъектов L'_o из L_o, по отношению к которым субъект u имеет право на операцию «Просмотр».

$$L'_o = \cup\{o\}$$

$$\forall o \in Lo, op = \text{‘Просмотр’}:$$

$$\delta a(u, o, op)=1$$

2. Определим список операций L_{op}, которые субъект u может осуществить над объектами из списка L'_o.

$$L_{op} = \cup \{ op \}$$
$$\forall op \in OP, \forall o \in L'o:$$
$$\delta a(u , o , op) = 1$$

Заключение

Рассмотренный в работе подход успешно апробирован на практике при создании механизмов разграничения доступа для систем электронного документооборота производственного назначения. Его применение показало адекватность описательных возможностей предложенной математической модели для представления семантики средств управления доступом в системах документооборота, а также возможность построения на ее основе надежных и практичных средств разграничения доступа, учитывающих свойства жизненного цикла и процесса маршрутизации документов, динамику функционирования систем.

Литература

1. Грушо А.А., Тимонина Е.Е. "Теоретические основы защиты информации" – издательство "Яхтсмен", Москва 1996г.
2. "Trusted Computer System Evaluation Criteria" ("The Orange Book") – DoD 5200.28-STD, CSC-STD-001-83.
3. Harrison, M.H., Ruzzo, W.L. and Ullman, J.D.Protection in Operating Systems." Communications of ACM 19(8):461-471 (1976).Grant Model." Journal of Computer and Systems Sciences 23(3):337-347 (1981).
4. R.S. Sandhu. The typed access matrix model. In Proceedings IEEE Computer Society Symposium on Research In Security and Privacy, pages 122{136, Oakland, CA, May 1992.
5. J.-J. Hwang, B.-M. Shao and P.C. Wang. A New Access Control Method Using Prime Factorisation. The Computer Journal, Vol. 35, No. 1, 1992.
6. L.J. LaPadula. Rule-Set Modeling of a Trusted Computer system. Information security: an integrated collection of essays / [edited by] Marshal Abrams, Sushil Jajodia, Haroll J. PodellQA76.9A25I5415, 1995.

Брусенцов Н.П.

Парадоксы логики, здравый смысл и диалектический постулат Гераклита-Аристотеля

На протяжении без малого ста лет предпринимаются настойчивые попытки преодолеть парадоксы импликации – характеристической функции лежащего в основании логики отношения следования. Неадекватность содержательному следованию ее "классической" двухзначной разновидности – материальной импликации – обусловила появление строгой, сильной, казуальной, релевантной, коннексивной и ряда трехзначных импликаций. Но проблема так и не решена, а обилие "результатов" свидетельствует о ее запутанности [1, с. 98].

В силлогистике отношение следования представлено общеутвердительной посылкой: "Всякое x есть y", сущность которой Аристотель исчерпывающе охарактеризовал в "Первой аналитике" [2, с. 215, 57в1]:

"...когда два [объекта] относятся друг к другу так, что если есть один, необходимо есть и второй, тогда, если нет второго, не будет и первого; однако если второй есть, то не необходимо, чтобы был и первый. Но невозможно, чтобы одно и то же было необходимо и когда другое есть, и когда его нет."

Здесь "первый" соответствует термину x, а "второй" – термину y. То, что при наличии первого необходимо есть (не может не быть) и второй, означает существование xy-вещей и несуществование xy'-вещей, а то, что при отсутствии второго не будет и первого, - существование $x'y'$-вещей и, еще раз, несуществование xy'-вещей. То же, что при наличии второго не необходимо, чтобы был и первый, не исключает возможности существования $x'y$-вещей, не настаивая на необходимости его. Невыразимостью именно этого *привходящего* статуса $x'y$-вещей в двухзначных исчислениях объясняется безуспешность всех попыток адекватно отобразить в них отношение содержательного следования.

Употребив в качестве символа существования вещей (т. е. принадлежности их множеству взаимосвязанных рассматриваемым отношением) префиксную интегральную дизъюнкцию (дизъюнкт) Vx – "существуют x-вещи", нетрудно убедиться, что аристотелеву отношению следования соответствует нечеткое множество

$$VxyV'xy'Vx'y'$$

Ему необходимо принадлежат *xy*- и *x'y'*-вещи, антипринадлежат *xy'*-вещи, а умолчание принадлежности *x'y*-вещей означает ее привходящий статус – *x'y*-вещи могут принадлежать либо антипринадлежать, существовать либо не существовать. Дизъюнкт V*x* – синоним льюисова ◊*x* [1, с. 12] и модальной функции M*x*, интерпретируемых обычно как "Возможность *x*", что не вполне корректно, поскольку надо различать актуальную возможность (существование) и потенциальную возможность (неисключенность – может существовать, а может не существовать). Вот эту выражаемую умолчанием неисключенность естественней называть просто "возможностью" в противоположность "существованию" и "несуществованию".

Нечеткое множество, "овеществляющее" посылку "Всякое *x* есть *y*" представимо с учетом сказанного следующими выражениями:

$$\mathrm{V}xy\mathrm{V}'xy\,'\mathrm{V}x'y' \equiv \mathrm{V}x\mathrm{V}'xy\,'\mathrm{V}y' \equiv \Diamond x\Diamond' xy'\,\Diamond y' \equiv \mathrm{M}x\mathrm{M}'xy\,'\mathrm{M}y'$$

Небезынтересно заметить, что строгой импликации Льюиса ◊'*xy*' для того, чтобы стать непарадоксальным следованием, недостает ◊*x*◊*y*' – утверждения о существовании *x*- и *y*'-вещей. И надо сказать, что это не льюисово упущение, а принципиальный изъян современной логики в целом, причем возникший не самопроизвольно [3, с. 79]:

"...по Аристотелю, высказывание "Все *A* суть *B*" считается истинным, лишь если существуют предметы, которые суть *A*. Наше отклонение в этом пункте от Аристотеля оправдывается потребностями математических применений логики, где класть в основу аристотелево понимание было бы нецелесообразно".

Вот так, под предлогом математических применений (на самом деле вследствие двухзначности "классических" исчислений) утратили здравый смысл.

Естественноязыковый смысл слов "Все *A* суть *B*" ("Всякому *A* присуще *B*", "*B* содержится в *A*"), выражающих содержательное, необходимое следование, свели к парадоксальной льюисовой импликации V'*AB*', удовлетворяющейся даже на пустом множестве вещей. При таком "следовании" оказывается, что "Из противоречия (из лжи) следует все, что угодно", и Лукасевич алгебраически доказывает [4, с. 94], что Аристотель ошибся, утверждая, что "невозможно, чтобы одно и то же было необходимо и когда другое есть, и когда его нет".

Стойким приверженцем здравомыслия выступает Льюис Кэррол [5], отличающий "Все *x* суть *y*" от "Ни один *xy*' не существует". Первое суждение он трактует как V*x*V'*xy*', а второе как V'*xy*', полагая, что в первом содержится утверждение о существовании *x*-вещей, а во втором нет. Чтобы стать аристотелевым контрапозитивным

следованием, кэрролову $V x V' xy'$ недостает $V y'$, т. е. проблема почти решена – изыскан смысловой, неформальный подход. Но этот актуальнейший результат Кэррола и его восхитительная трехзначная диаграмма уже более ста лет остаются невостребованными. Совершенно очевидна справедливость его беспощадного юмора: "Я буду называть их "логиками", надеясь, что в этом нет ничего обидного" [5, с. 329].

Вследствие допущенного "отклонения" в современных логических исчислениях невыразима силлогистика Аристотеля. Это другая проблема, которую тщетно пытаются решить логики, а вернее, другая сторона той же проблемы содержательного следования. Силлогистика, вопреки всеобщему убеждению, не удовлетворяет закону исключенного третьего. Она трехзначна, в основу ее положен несовместимый с исключением третьего диалектический принцип – постулат Гераклита-Аристотеля [6], утверждающий *сосуществование противоположностей*. Алгебраически он выражается тождеством:

$$V x V x' V y V y' V z V z' \ldots \equiv 1.$$

Математический смысл его в том, что первичные (несоставные) термины силлогистики x, y, z, ... не могут быть безусловными константами, а должны представлять собой переменные, принимающие взаимоисключающие значения согласно гераклитову "все течет, все изменяется". Существование x-вещей мыслимо лишь как сосуществование противоположностей – $V x V x'$.

Приведенные выше дизъюнктивные выражения представляют отношение следования как нечеткое множество взаимосвязанных вещей. Отношение материальной импликации $x \rightarrow y$ в таком представлении будет $V xy V xy' V x' y V x' y'$. Обычно используют его характеристическую функцию: $x' \vee y$. Функция, характеризующая содержательное следование в таком смысле, оказывается трехзначной: $xy \vee \sigma x' y \vee x' y'$, где σ - третье значение истинности, $0 < \sigma < 1$.

Литература

1. Слинин Я.А. Современная модальная логика. – Л: Изд-во Ленингр. ун-та, 1976.
2. Аристотель. Сочинения в четырех томах. Том 2. – М: "Мысль", 1978.
3. Гильберт Д., Аккерман В. Основы теоретической логики. – М: ИЛ, 1947.

4. Лукасевич Я. Аристотелевская силлогистика с точки зрения современной формальной логики. – М.: ИЛ, 1959.

5. Кэррол Л. Символическая логика // Льюис Кэррол. История с узелками. – М.: "Мир", 1973.

6. Брусенцов Н.П. Интеллект и диалектическая триада // "Искусственный интеллект", 2'2002, - Донецк, 2002. С. 53-57.

Раздел II
Машинная графика и обработка образов

Нековаль С.П.
Метод фазовой корреляции решения задачи анализа движения в видеоизображениях

Введение

Задача анализа движущихся фрагментов видеоизображения вызывает большой интерес у специалистов в области обработки видео. Представление видеоизображения в виде набора движущихся объектов позволяет применять нетривиальные методы обработки. Это представление также оказывается удобным для преобразования временных характеристик видеоизображения.

Существует целый ряд задач, базирующийся на анализе движения. Это, прежде всего, задача видеокомпрессии. В этом случае анализ движущихся фрагментов позволяет выделить свойственную видеоизображениям избыточность. В большинстве случаев кадры видеоизображения слабо меняются с течением времени. Анализ движения объектов в кадре позволяет уловить эти изменения. В дальнейшем эта информация может быть использована для компрессии. Особая точность анализа здесь не требуется, однако может положительно повлиять на степень сжатия.

Другая область применения – преобразование стандартов видеоизображений. В мире существует множество различных телевизионных и видеостандартов, зачастую имеющих не только разные параметры развертки (число строк), но и различную частоту кадров. Изменение частоты кадров в видеоизображении – процесс нетривиальный, и качественная его реализация затруднительна без учета движущихся объектов. [1] Здесь очень важна точность поиска этих объектов, поскольку от него напрямую зависит качество результирующего изображения.

Результаты анализа также с успехом используются при преобразовании телевизионного сигнала в прогрессивный формат развертки (*de-interlace*). Анализ движения позволяет избежать неприятных эффектов, возникающих при быстрых перемещениях объектов в кадре. [3]

Данная статья посвящена одному из наиболее перспективных методов анализа движения – методу фазовой корреляции.

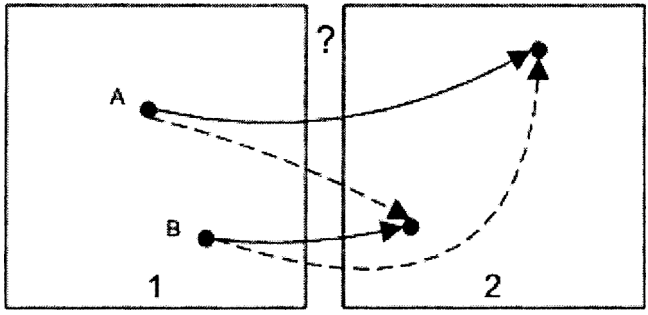
1. Основные понятия

Отметим сразу, что в качестве исходных данных рассматривается разбитая на отдельные кадры видеопоследовательность. Все кадры имеют одинаковый размер и представлены как двумерный набор отдельных точек (пикселов). Следовательно, мы сразу ограничиваем область применения цифровыми видеоизображениями и отбрасываем такие, например, варианты, как изображение, отснятое на кинопленке, и телевизионное аналоговое изображение. Пиксельное представление кадра позволит в дальнейшем применять переборные алгоритмы. Без дискретизации проведение анализа будет весьма и весьма затруднительным.

Опишем подробнее, что же, собственно, является результатом анализа. В большинстве случаев анализируется движение между двумя соседними кадрами. Это, однако, не препятствует алгоритму использовать при анализе все остальные кадры. Результатом является набор так называемых **векторов движения** (motion vectors). Каждый вектор указывает направление движения обособленной группы точек (назовем эту группу точек **объектом**). Итак, алгоритм анализа выделяет в исходном кадре некие объекты и сопоставляет каждому направление движения, происходящего при переходе к следующему кадру. При этом на следующем кадре присутствует **образ** движущегося объекта; будем говорить, что этот образ в некотором смысле *похож* на сам объект. Понятие *похожести* в каждом алгоритме определяется по-своему, неформально можно понимать это как некую смысловую связь между кадрами.

К сожалению, очень редко кадры выглядят таким образом, что «все объекты куда-то движутся». Реальные видеоизображения настолько сложны, что могут существовать, например, объекты, не имеющие аналогов в следующем кадре (здесь подразумевается, что алгоритм не смог найти образ такого объекта при движении); такие объекты могут интерпретироваться как исчезнувшие. Объекты могут и возникнуть «ниоткуда». В работе [5] рассматривается также понятие **монтажного плана** (МП). Монтажный план – понятие, пришедшее из кинофильмов и означающее некоторый сюжетно ограниченный временной сегмент фильма; весь фильм, таким образом, есть совокупность склеенных вместе монтажных планов. Смена монтажного плана действительно частое явление в кинематографе, однако прогресс постоянно стремится к тому, чтобы сгладить этот момент «склейки», сделать его более плавным, тем самым корректное определение границ монтажного плана становится все более и более сложным.

Принципиальная проблема состоит в невозможности выработки единого критерия оценки качества алгоритмов анализа. Затруднение это можно продемонстрировать следующим примером:



Мы видим в первом кадре две одинаковые точки A и B. Следующий кадр можно проинтерпретировать следующим образом: точки A и B сместились в другое положение. Образы каких именно точек мы видим на втором кадре? На этот вопрос однозначно ответить нельзя; можно дать сразу два варианта ответа, поскольку точки A и B абсолютно одинаковы. Такая ситуация, хотя она и далека от реальности, указывает на принципиальную неразрешимость задачи анализа. Это говорит о том, что формально установить достоверность результата невозможно. Окончательная оценка качества всегда остается за человеком.

2. Базовые сведения

Большинство алгоритмов анализа движущихся фрагментов являются по сути переборными. Действительно, дискретное представление видеоизображения делает такой перебор возможным. Это, например, метод блочного сравнения, представляющий собой наиболее прямолинейную реализацию перебора: проверить все возможные группы точек и все возможные векторы движения.

Метод фазовой корреляции не исключение. Он также является переборным. Однако использование преобразования Фурье позволяет сильно сократить перебор. Основная идея метода – тот факт, что сдвиг в пространстве эквивалентен фазовому сдвигу в частотной области. Оказывается, что в данном случае удобнее сначала представить кадры видеоизображения в виде сумм ряда Фурье.

3. Преобразование Фурье

Преобразование Фурье позволяет представить входной сигнал в виде зависимости амплитуды от частоты. Существуют модификации этого преобразования для функций многих переменных.

Преобразование может быть применено и для дискретного случая. В этом случае исходная функция задается набором своих значений на сетке.

Пусть $f_{k,n}$ (k, n = 0,1,..., N-1) – исходная дискретная функция двух переменных. Для простоты будем предполагать, что она задана в квадрате на однородной сетке. Результат дискретного преобразования Фурье (*спектр*) выглядит следующим образом:

$$\hat{f}_{u,v} = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{n=0}^{N-1} f_{k,n} \exp\left\{-2\pi i \frac{ku+nv}{N}\right\}, \; u,v=0,1,...,N\text{-}1, \qquad (1)$$

где $\hat{f}_{u,v}$ (u,v=0,1,..., N-1) - коэффициент Фурье исходной двумерной дискретной функции. Здесь u,v – *частоты*.

Заметим, что $\hat{f}_{u,v}$ - результат выражения (1) - есть двумерная дискретная комплекснозначная функция. Исходная же функция $f_{k,n}$ принимает только действительные значения (интенсивность соответствующей точки в кадре). Каждый коэффициент Фурье может быть представлен в *полярных координатах*: $\hat{f}_{u,v} = A_{u,v}e^{i\varphi_{u,v}}$.

Коэффициент $A_{u,v}$ называется *амплитудой*, а $\varphi_{u,v}$ - *фазой*.

Обратное преобразование осуществляется по формуле:

$$f_{k,n} = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} \hat{f}_{u,v} \exp\left\{2\pi i \frac{ku+nv}{N}\right\}, \; k,n=0,1,\ldots,N\text{-}1, \qquad (2)$$

Отметим следующее свойство коэффициентов Фурье:

$$\hat{f}^{(1)}_{u,v}\hat{f}^{(2)}_{u,v} = A^{(1)}_{u,v}A^{(2)}_{u,v}e^{i(\varphi^{(1)}_{u,v}+\varphi^{(2)}_{u,v})} \qquad (3)$$

Иначе говоря, при умножении спектров амплитуды, соответствующие одинаковым частотам, перемножаются, а фазы – складываются. Отсюда нетрудно получить следующую формулу:

$$\hat{f}^{(1)}_{u,v}\hat{f}^{(2)*}_{u,v} = A^{(1)}_{u,v}A^{(2)}_{u,v}e^{i(\varphi^{(1)}_{u,v}-\varphi^{(2)}_{u,v})} \qquad (4)$$

Здесь * означает комплексно-сопряженную величину.

4. Преобразование фаз

Пусть имеются два соседних кадра видеоизображения. Как и ранее, считаем, что изображения заданы в цифровом (дискретном) виде на равномерной сетке. Ограничимся изображениями в градациях серого, где значение соответствующей дискретной функции $f_{k,n}$ определяет интенсивность серого цвета в каждой точке (k,n). Пусть $\hat{f}_{u,v}$ – дискретное преобразование Фурье для функции $f_{k,n}$ (где u,v – частоты, а k,n – координаты). Рассмотрим следующее преобразование:

$$\Phi\left[f^{(1)}_{k,n}, f^{(2)}_{k,n}\right] = \frac{\hat{f}^{(1)*}_{u,v}\hat{f}^{(2)}_{u,v}}{\left|\hat{f}^{(1)*}_{u,v}\hat{f}^{(2)}_{u,v}\right|} \qquad (5)$$

Здесь $f^{(1)}_{k,n}$ и $f^{(2)}_{k,n}$ – функции, задающие первый и второй кадр соответственно, а $\hat{f}^{(1)}_{u,v}$ и $\hat{f}^{(2)}_{u,v}$ - их дискретные преобразования Фурье, звездочка означает комплексно-сопряженную величину.

Выражение это, по сути, представляет собой нормализованный *кросс-спектр*, т.е. преобразование Фурье от кросс-корреляции функций $f^{(1)}_{k,n}$ и $f^{(2)}_{k,n}$. Кросс-корреляция позволяет описать степень зависимости этих двух функций. Заметим, что фактически формула (5) есть нормализованный вариант формулы (4). Нормализация в данном случае означает, что из результирующего сигнала $\Phi\left[f^{(1)}_{k,n}, f^{(2)}_{k,n}\right]$ будет полностью исключена информация об амплитудах, т.е.

$$\Phi\left[f^{(1)}_{k,n}, f^{(2)}_{k,n}\right] = e^{i(\varphi^{(2)}_{u,v}-\varphi^{(1)}_{u,v})} \qquad (6)$$

Следующим этапом преобразования является перевод сигнала обратно в пространственную область. Для этого к $\Phi\left[f^{(1)}_{k,n}, f^{(2)}_{k,n}\right]$ применяется обратное дискретное преобразование Фурье.

Результат преобразования есть дискретная функция двух переменных, определяющая некоторую поверхность в той же квадратной области, что и исходные кадры.

Поверхность эта, называемая ***корреляционной***, позволяет судить о движении объектов между исходными двумя кадрами. Рассмотрим, как движение объектов между кадрами влияет на форму поверхности.

5. Корреляционная поверхность

Можно отметить несколько особенностей корреляционной поверхности. Прежде всего, рассмотрим три простейших варианта:

1. Отсутствие движения

Принимая во внимание тождество

$$f_{k,n}^{(2)} = f_{k,n}^{(1)},$$

в результате преобразования получаем:

$$\Phi\left[f_{k,n}^{(1)}, f_{k,n}^{(2)}\right] = \frac{\hat{f}_{u,v}^{(1)*}\,\hat{f}_{u,v}^{(1)}}{\left|\hat{f}_{u,v}^{(1)*}\,\hat{f}_{u,v}^{(1)}\right|} = 1$$

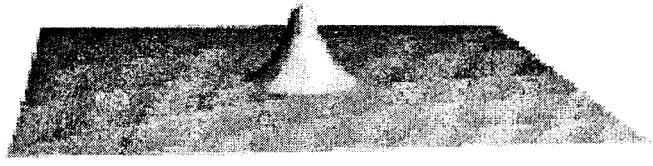


Рисунок 1. Отсутствие движения

Обратное преобразование Фурье от константы представляет собой функцию, принимающую ненулевое значение только в одной точке. При этом поверхность состоит из единственного пика в центре (условно назовем *центром* точку с координатами (0, 0)).

2. Поступательное движение

Допустим, один кадр сдвинулся относительно другого на *(Δk,Δn)*, т.е. между функциями $f_{k,n}^{(1)}$ и $f_{k,n}^{(2)}$ имеется следующее соотношение:

$$f_{k,n}^{(2)} = f_{k+\Delta k,n+\Delta n}^{(1)}$$

Нетрудно получить формулу:

$$\Phi\left[f_{k,n}^{(1)}, f_{k,n}^{(2)}\right] = e^{2\pi i \frac{u\Delta k + v\Delta n}{N}}$$

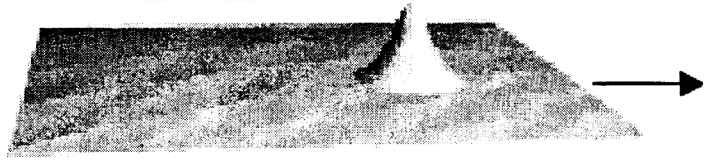


Рисунок 2. Поступательное движение

Один поступательно движущийся объект порождает смещение центрального пика в горизонтальной плоскости на величину, соответствующую пройденному расстоянию.

3. Вращательное движение

Исходим из соотношения

$f_{k,n}^{(2)} = f_{k,n}^{(1)} \cdot e^{i\varphi}$ (поворот на угол φ).

С точки зрения амплитудно-фазового представления функции $f_{k,n}^{(2)}$ это означает, что каждая из фаз $\varphi_{u,v}$ изменится на величину φ. При этом амплитуды $A_{u,v}$ остаются неизменными. Отсюда получаем:

$$\Phi\left[f_{k,n}^{(1)}, f_{k,n}^{(2)}\right] = e^{i\varphi}$$

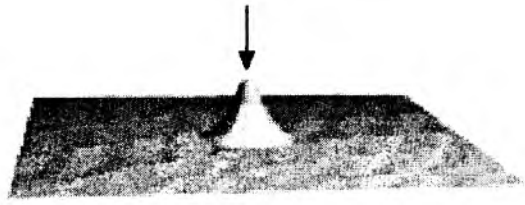


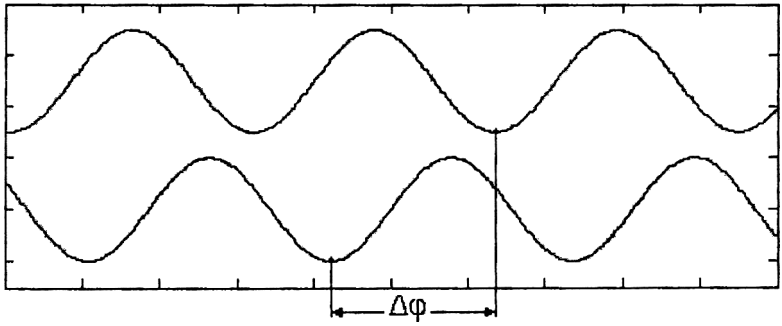
Рисунок 3. Вращательное движение

Результатом обратного преобразования Фурье от константы $e^{i\varphi}$ будет все тот же пик в центре, однако на этот раз с ненулевой комплексной составляющей и с высотой, отличной от единицы (в отличие от случая 1).

Итак, вращение камеры порождает изменение высоты центрального пика, соответствующее углу поворота.

При наличии нескольких движущихся объектов ситуация усложняется. На поверхности появляется множество побочных «всплесков». Поэтому следующим этапом метода фазовой корреляции будет анализ пиков корреляционной поверхности. Установив положение пика, можно найти соответствующий вектор движения (случай 2). Найдя все пики, мы найдем все векторы движения.

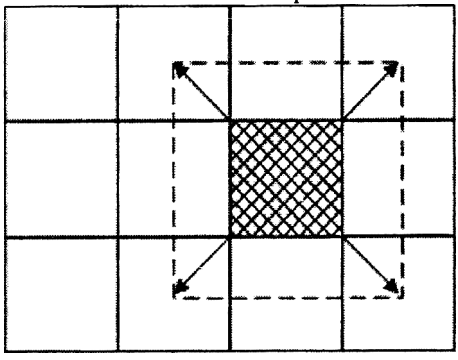
Формула (6) показывает, что построение корреляционной поверхности в действительности суть расчет сдвига одного кадра по фазе относительно другого с последующим переходом в пространственную область. Сдвиг по фазе же эквивалентен пространственному сдвигу, что можно продемонстрировать следующим примером (для одномерного случая):



Приведенный график демонстрирует фазовый сдвиг для одной гармоники. В простейшем случае пространственный сдвиг функции на Δx означает сдвиг каждой гармоники по фазе на тот же самый Δx.

6. Окна

Вычислительная сложность преобразования Фурье существенно увеличивается с увеличением размера изображения. Поэтому естественным образом возникает стремление уменьшить обрабатываемую область; это можно сделать, разделив весь кадр на несколько более мелких прямоугольных участков (называемых *окнами*) [4]. Рассчитать корреляционную поверхность для каждого из этих участков легче, чем провести обработку всего кадра целиком. Однако при этом мы сознательно ограничиваем область поиска векторов движения. Действительно, поскольку поверхность рассчитывается только для прямоугольного окна малого размера, длина полученных векторов всегда не превышает половины размера этого окна (это следует из структуры корреляционной поверхности). Разбиение на окна позволяет ускорить вычисление, однако при этом теряются все «быстрые» движения. Такое разбиение действительно необходимо, но размер окна нужно подбирать как компромисс между сложностью вычислений и точностью распознавания движения.



Следует отметить, что каждая точка окна также может порождать векторы движения в соседних окнах. Корректная реализация алгоритма должна уметь отбрасывать ненужные векторы. Соответствующую методику можно найти в книге [4].

7. Поиск движущихся объектов

Итак, преобразование Фурье и анализ корреляционной поверхности позволяет выделить все имеющиеся векторы движения. Однако корреляционная поверхность не предоставляет никакой информации о том, каким именно точкам в кадре соответствуют эти векторы. Именно для этого введен такой этап алгоритма, как сопоставление векторов и движущихся точек.

Пусть известно, что какие-то точки кадра движутся в заданном направлении. Как можно найти эти точки? Учтем, что движение может происходить только в пределах окна.

Необходимо найти, какие точки в кадре были сдвинуты на заданный вектор. Один из подходов – сдвинуть один из кадров на этот вектор и сравнить с другим кадром. Те из точек кадра, которые совпадут, можно считать сопоставленными с данным вектором движения.

«Совпадение» здесь следует трактовать в широком смысле (совпадение с некоторой точностью). Полное совпадение редко присутствует в реальном видеоизображении. Собственно, именно критерии совпадения и будут формально определять понятие *объекта* в конкретной реализации алгоритма.

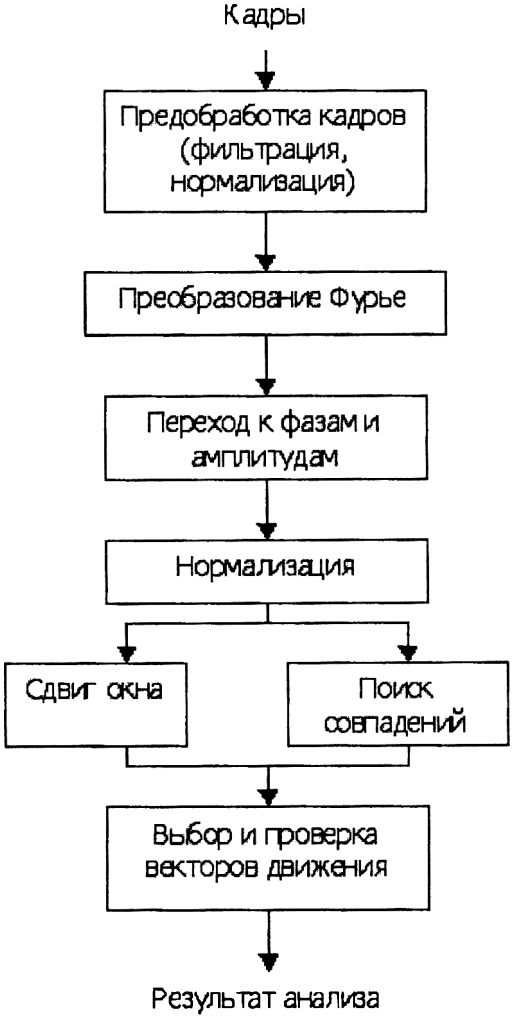
Итак, проводить точное пиксельное сравнение внутри окна нет смысла. Поэтому используются приближенные критерии, в частности среднеквадратичное отклонение (MSE), минимальное абсолютное отклонение (MAD) и суммарное абсолютное отклонение (SAD). Предел точности (величина ошибки, при достижении которой считается, что совпадение произошло – *threshold*) подбирается экспериментально и является одним из параметров алгоритма.

8. Субпиксельная точность

Заметим, что результат выражения для кросс-корреляции (3) является, вообще говоря, дробным числом. Отбрасывая дробную часть и тем самым рассматривая только движения на целое число пикселов, мы несколько огрубляем результат анализа. Оказывается, что можно существенно улучшить качество, рассматривая движения на некоторые доли пикселов. Одним из распространенных вариантов является *полупиксельная точность*, когда векторы движения рассчитываются с точностью до полупиксела. В этом случае можно избежать медленных вычислений с плавающей точкой и использовать быструю целочисленную арифметику. Недостающие «половинные» отсчеты на корреляционной поверхности можно найти интерполяцией, например, сплайнами. Практика показывает, что даже полупиксельные вычисления позволяют существенно улучшить результат и снизить случайный шум, возникающий от неточностей поиска векторов движения. [2]

9. Основные этапы алгоритма

Алгоритм анализа, основанный на методе фазовой корреляции, можно представить в виде набора следующих этапов:



Промежуточная нормализация полностью подавляет информацию об амплитудах, после чего анализируются только фазы.

Для обработки ТВ-кадров алгоритм существенно усложняется. Это связано с необходимостью правильно обрабатывать чересстрочный формат развертки изображения.

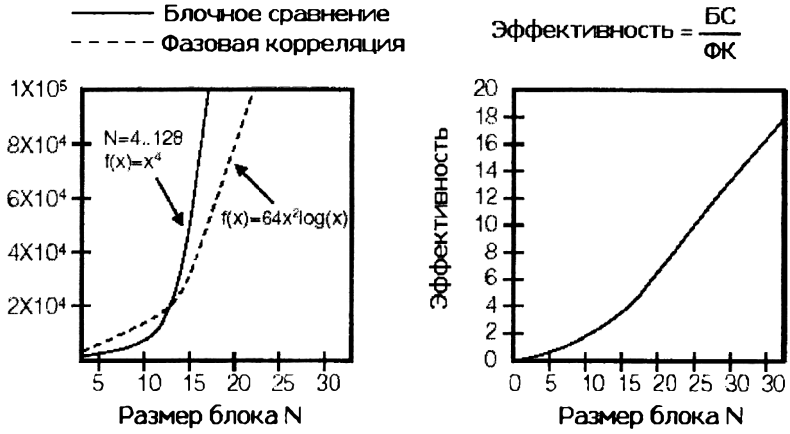
10. Сравнительная характеристика метода фазовой корреляции

В заключении рассмотрим оценку результатов работы метода фазовой корреляции и метода, являющегося одним из типичных представителей переборного алгоритма анализа движения.

Блочное сравнение (БС) – один из наиболее известных алгоритмов поиска движущихся фрагментов. Смысл алгоритма вкратце сводится к следующему. Кадр А делится на прямоугольные блоки заранее оговоренного фиксированного размера. Рассмотрим некоторый блок кадра и соответствующий ему блок соседнего кадра Б (находящийся в той же позиции). Если соответствующие участки близки по содержанию (*коррелируют*), можно говорить об отсутствии движения внутри выбранного блока. Если движение все-таки присутствует (фрагменты не коррелируют), попытаемся найти рассматриваемый блок внутри соседнего кадра Б, сдвигая его на различные позиции. Аналогичным образом этот поиск производится для каждого блока исходного кадра А.

Алгоритм блочного сравнения является наиболее простым переборным алгоритмом. Полный перебор требует огромного количества вычислений. Поэтому область поиска пытаются ограничить. Различные варианты алгоритма блочного сопоставления отличаются именно способами упростить вычисления.

Как уже отмечалось, в методе фазовой корреляции перебора также не избежать. Однако знание векторов движения объектов позволяет существенно упростить задачу. Важно то, что вычислить дискретное преобразование Фурье легче, нежели перебрать все возможные векторы движения. Этому способствует алгоритм *быстрого преобразования Фурье* (БПФ), позволяющий существенно сократить число операций при вычислении спектра функции.



Метод фазовой корреляции заметно лучше справляется с регулярными структурами (повторяющаяся последовательность одинаковых объектов, например, решетка). Блочное сравнение не позволяет корректно отличать повторяющиеся объекты и тем самым порождает значительные искажения.

В то же время метод фазовой корреляции плохо приспособлен для поиска движения в окнах малого размера. Уменьшение области поиска сужает частотный спектр и ухудшает разрешающую способность алгоритма. В работе [7] приблизительный минимальный размер окна указан как 64x64. Там же приведена специальная модификация алгоритма для окон малого размера.

Заключение

Несмотря на относительную простоту формулы (3), практическая реализация метода фазовой корреляции является весьма нетривиальной задачей. Особенно внимательно надо отнестись к выбору параметров – таких как размер окна, степень точности при приближенном сравнении окон и т.п. Однако потенциал метода очень высок; тот факт, что анализ пиков корреляционной поверхности позволяет сразу выделить преобладающее движение в кадре, существенно повышает точность результата. Использование субпиксельных вычислений также дает существенный прирост в качестве.

Метод фазовой корреляции с успехом используется в различных телевизионных преобразователях стандартов с компенсацией движения (*MC, motion compensated*). Метод как нельзя лучше подходит для этих целей, распознавая самые сложные виды движения, справляясь

и с такими проблематичными ситуациями, как наличие регулярных структур, движение объекта на сложном для распознавания фоне и т.п. В приложениях, не столь критичных к точности анализа движения (например, компрессия), можно использовать более простые в реализации блочные или градиентные методы.

Интенсивное изучение и дальнейшее усовершенствование метода фазовой корреляции проводится Philips и BBC Research Labs. [1] Один из лучших на сегодняшний день телевизионных преобразователей стандартов – Alchemist Ph.C. компании Snell & Wilcox – использует фазовую корреляцию в качестве базового алгоритма своей работы. [3]

Литература

1. Marsh D. Temporal Rate Conversion. – Microsoft, 2001
2. Yi Liang. Phase-Correlation Motion Estimation. – Stanford ISE, 2000
3. Watkinson J. The Engineer's Guide to Standards Conversion. – Snell & Wilcox, 1994
4. Watkinson J. The Engineer's Guide to Motion Compensation. – Snell & Wilcox, 1994
5. Казаков Я.В. Повышение качества киноизображения средствами вычислительной техники. - СПбГУ кино и телевидения, кафедра кино-видеоаппаратуры, 2003
6. Лукин А. Введение в цифровую обработку сигналов. - Лаборатория компьютерной графики и мультимедиа, МГУ, 2002
7. Hill L., Vlachos T. Shape Adaptive Phase Correlation. - IEE Electronics Letters, Vol. 37, No. 25, pp1512-1513, 2000

Подшивалов А.Ю.

Определение пространственного расположения 3D объектов по маркерам

Работа посвящена проблеме создания алгоритмов определения пространственного расположения объектов, имеющих большие размеры и протяженность вдоль нескольких произвольно ориентированных осей (таких как современные и перспективные долговременные орбитальные станции и их сегменты). Данную задачу необходимо решать на каждом шаге работы системы моделирования. 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 точности вычислений в связи с высоким риском столкновений объектов. В работе предлагается подход с использованием высокоточных маркеров, привязанных к объектам. При этом в систему моделирования поступает два вида информации: априорная информации о поверхностях объектов, и апостериорная - о координатах маркеров. Показано, что такой подход позволяет эффективно решать задачу для 3D-объектов высокой сложности.

Введение

Одной из основных задач систем Виртуальной Реальности при моделировании физической системы является задача определения текущего пространственного расположения объектов этой системы. Поскольку работа системы моделирования выполняется итерационно, данную задачу необходимо решать многократно, на каждой итерации, с использованием последних поступивших данных о состоянии физической системы. Для получения информации от физической системы часто используются так называемые «системы технического зрения». Вообще говоря, существует множество подходов к реализации систем технического зрения, большинство из которых основаны на анализе набора проекций изображений трехмерной среды [1, 2]. Такие подходы трудоемки, поскольку требуется в той или иной степени решать задачи распознавания образов, такие как выделение характерных фрагментов, фильтрация и т.д.

При моделировании поведения долговременных орбитальных станций и космических аппаратов сеточные модели объектов содержат

порядка 10^7-10^8 узлов. Кроме того, такие объекты имеют сложную пространственную конфигурацию (наличие протяженных конструкций, расположенных вдоль нескольких осей). При движении объектов на расстояниях, соизмеримых с их характерными размерами, необходимо следить за большим количеством узлов моделей объектов, а это, в свою очередь, означает еще большее увеличение сложности задачи определения расположения объектов.

Заметим, что рост мощностей вычислительных комплексов в последние годы носит экспоненциальный характер, и подобные задачи могут быть решены даже в режиме реального времени с использованием наземных ВЦ. Однако мы рассматриваем задачу в контексте систем автономного управления, а это означает, что задача должна быть решена бортовыми системами. Таким образом, необходимо искать пути как упрощения постановки задачи, так и повышения эффективности используемых алгоритмов.

Возможности технологии «Виртуальная реальность» позволяют использовать метод «перехват движений» (Motion Capture) [3] как основу для определения расположения объектов. В основе метода лежит слежение за относительно небольшим количеством опорных точек.

В последнее время подход «Motion capture» широко используется в кинематографии для имитации движений реального человека с помощью компьютерной модели. Для этого к человеку в ключевых точках прикрепляются специальные маркеры. При перемещении человека сенсорная подсистема снимает координаты маркеров, и на их основе воссоздает все движения человека. Отметим, что *качество синтезированного движения напрямую зависит от количества маркеров*. Например, в работах Andrea Bottino [4] для имитации движения человека используются 3D-координаты маркеров вместе с набором двухмерных проекций. Решение задачи основано на том факте, что точки объекта за время одной итерации смещаются на очень малые расстояния. Тем не менее, по оценкам самих авторов сложность алгоритма носит квадратичную зависимость от линейных размеров модели, что неприемлемо при рассмотрении объектов типа космических станций. Кроме того, требуется устранять ошибки, связанные с возникновением точек-фантомов (используется фильтрация).

В нашем случае предполагается использование трехмерных сеточных моделей объектов, построенных априорно. Это означает, что рассмотренные выше подходы обладают высокой избыточностью, поскольку большой процент информации о расположении объектов уже содержится в их моделях. Объемы регистрируемой информации и сложность используемых алгоритмов могут быть радикально понижены, поскольку необходимо лишь определить пространственные

координаты ограниченного набора точек объекта (как минимум, трех точек, не лежащих на одной прямой).

1. Постановка задачи.

В рассматриваемой системе моделирования физические объекты представлены с помощью их сеточных моделей, построенных априорно. За объектами физической системы закрепляются маркеры, а подсистема слежения, в свою очередь, позволяет определять их координаты. Таким образом, задача, рассматриваемая в данной работе, состоит в определении пространственного расположения объектов на основе информации о трехмерных координатах маркеров (и априорной информации о моделях объектов).

Рассмотрим две системы координат (СК): мировую СК и СК, привязанную к данному объекту. Для того, чтобы определить пространственное расположение объекта в мировой СК, достаточно определить матрицу преобразования из СК объекта в мировую СК. Тогда, имея априорную информацию о координатах узлов сеточной модели в СК объекта, мы сможем определить координаты любого узла в мировой СК.

Вообще говоря, для построения матрицы преобразования достаточно знать координаты любых трех маркеров, не лежащих на одной прямой. Однако на практике количество маркеров может быть существенно большим, а координаты маркеров определяются с некоторой погрешностью (погрешность зависит от многих факторов, в первую очередь от параметров подсистемы слежения). Поэтому нам необходимо определить матрицу преобразования так, чтобы невязка между вычисленными и реальными координатами маркеров была минимальной.

Итак, требуется на основе известных координат маркеров и априорной информации о моделях объектов построить матрицу преобразования координат из СК объекта в мировую СК, с учетом минимизации невязки.

2. Алгоритм решения.

Рассмотрим произвольный объект моделируемой сцены. Пусть подсистема слежения определила 3D-координаты n маркеров, привязанных к данному объекту. Будем искать коэффициенты матрицы преобразования, заданной в виде:

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Здесь необходимо сделать некоторые пояснения. Искомое преобразование является ортогональным (сохраняются расстояния и углы). Для описания трехмерных преобразований в компьютерной графике часто используют следующий подход. К трехмерным координатам векторов добавляют четвертую, которая всегда равна единице. Тогда с помощью матрицы размерностью 4x4 очень удобно описывать ортогональные преобразования: коэффициенты a_{11}-a_{33} описывают преобразование поворота, а коэффициенты a_{14},a_{24},a_{34} – сдвига.

За основу метода минимизации невязки возьмем метод наименьших квадратов [6]. Основания для выбора этого метода следующие:

1. простота метода;
2. квадрат расстояния между двумя точками идеально подходит в качестве меры погрешности.

Пусть мы определили коэффициенты матрицы преобразования. Тогда координаты i-го маркера $x_i^{реш}$, $y_i^{реш}$, $z_i^{реш}$, построенные с помощью найденного преобразования определяются как:

$$\left(x_i^{реш}, y_i^{реш}, z_i^{реш}, 1\right) = A * \left(x_i^{об}, y_i^{об}, z_i^{об}, 1\right)$$

где $x_i^{об}$, $y_i^{об}$, $z_i^{об}$ – координаты маркера в СК объекта (известны априорно). Невязка определения координат i-го маркера записывается в виде:

$$\rho_i^2 = \left(x_i^{реш} - x_i\right)^2 + \left(y_i^{реш} - y_i\right)^2 + \left(z_i^{реш} - z_i\right)^2 =$$
$$+\left(a_{11}\, x_i^{об} + a_{12}\, y_i^{об} + a_{13}\, z_i^{об} + a_{14} - x_i\right)^2 +$$
$$+\left(a_{21}\, x_i^{об} + a_{22}\, y_i^{об} + a_{23}\, z_i^{об} + a_{24} - y_i\right)^2 +$$
$$+\left(a_{31}\, x_i^{об} + a_{32}\, y_i^{об} + a_{33}\, z_i^{об} + a_{34} - z_i\right)^2$$

Таким образом, функция невязки для объекта в зависимости от коэффициентов матрицы преобразования записывается в виде:

$$f\left(a_{11}, a_{12}, a_{13}, a_{14}, a_{21}, a_{22}, a_{23}, a_{24}, a_{31}, a_{32}, a_{33}, a_{34}\right) = \sum_{i=1}^{n} \rho_i^2$$

Необходимым условием минимума функции в точке является равенство нулю всех частных производных в этой точке. Запишем частные производные функции f по всем аргументам (ниже приведены производные по аргументам a_{11}-a_{14}, производные по остальным аргументам вычисляются аналогично):

$$f'_{a11} = 2a_{11}\sum_{i=1}^{n}(x_i^{об})^2 + 2a_{12}\sum_{i=1}^{n}x_i^{об}y_i^{об} + 2a_{13}\sum_{i=1}^{n}x_i^{об}z_i^{об} + 2a_{14}\sum_{i=1}^{n}x_i^{об} - 2\sum_{i=1}^{n}x_i^{об}x_i$$

$$f'_{a12} = 2a_{11}\sum_{i=1}^{n}x_i^{об}y_i^{об} + 2a_{12}\sum_{i=1}^{n}(y_i^{об})^2 + 2a_{13}\sum_{i=1}^{n}y_i^{об}z_i^{об} + 2a_{14}\sum_{i=1}^{n}y_i^{об} - 2\sum_{i=1}^{n}y_i^{об}x_i$$

$$f'_{a13} = 2a_{11}\sum_{i=1}^{n}x_i^{об}z_i^{об} + 2a_{12}\sum_{i=1}^{n}y_i^{об}z_i^{об} + 2a_{13}\sum_{i=1}^{n}(z_i^{об})^2 + 2a_{14}\sum_{i=1}^{n}z_i^{об} - 2\sum_{i=1}^{n}z_i^{об}x_i$$

$$f'_{a14} = 2a_{11}\sum_{i=1}^{n}x_i^{об} + 2a_{12}\sum_{i=1}^{n}y_i^{об} + 2a_{13}\sum_{i=1}^{n}z_i^{об} + 2a_{14}n - 2\sum_{i=1}^{n}x_i$$

В итоге мы получаем систему из 12 линейных уравнений с 12-ю неизвестными. Заметим, что вся система распадается на 3 независимых системы для коэффициентов a_{11}-a_{14}, a_{21}-a_{24}, a_{31}-a_{34}. Например, для коэффициентов a_{11}- a_{14} система уравнений записывается в следующем виде:

$$a_{11}\sum_{i=1}^{n}(x_i^{об})^2 + a_{12}\sum_{i=1}^{n}x_i^{об}y_i^{об} + a_{13}\sum_{i=1}^{n}x_i^{об}z_i^{об} + a_{14}\sum_{i=1}^{n}x_i^{об} = \sum_{i=1}^{n}x_i^{об}x_i$$

$$a_{11}\sum_{i=1}^{n}x_i^{об}y_i^{об} + a_{12}\sum_{i=1}^{n}(y_i^{об})^2 + a_{13}\sum_{i=1}^{n}y_i^{об}z_i^{об} + a_{14}\sum_{i=1}^{n}y_i^{об} = \sum_{i=1}^{n}y_i^{об}x_i$$

$$a_{11}\sum_{i=1}^{n}x_i^{об}z_i^{об} + a_{12}\sum_{i=1}^{n}y_i^{об}z_i^{об} + a_{13}\sum_{i=1}^{n}(z_i^{об})^2 + a_{14}\sum_{i=1}^{n}z_i^{об} = \sum_{i=1}^{n}z_i^{об}x_i$$

$$a_{11}\sum_{i=1}^{n}x_i^{об} + a_{12}\sum_{i=1}^{n}y_i^{об} + a_{13}\sum_{i=1}^{n}z_i^{об} + a_{14}n = \sum_{i=1}^{n}x_i$$

Будем решать каждую из систем по правилу Крамера. Если случай невырожденный, и определители матриц систем линейных уравнений не равны нулю, мы получим единственное решение. Это

решение и будет точкой минимума функции невязки. Действительно, при стремлении аргументов к плюс-минус бесконечности функция невязки будет стремиться к плюс бесконечности, а значит, поскольку функция невязки является непрерывной и дифференцируемой во всех точках, в случае равенства нулю производной в единственной точке, эта точка должна являться точкой минимума.

Итак, получив решение системы уравнений, мы определили коэффициенты матрицы преобразования из СК объекта в мировую СК. Таким образом, для того чтобы получить вектор координат любого узла сеточной модели объекта в мировой СК $(x^{мир}, y^{мир}, z^{мир}, 1)$, необходимо умножить найденную матрицу преобразования A на вектор координат этого узла в СК объекта $(x^{об}, y^{об}, z^{об}, 1)$.

3. Условие существования и единственности решения.

Определение коэффициентов матрицы преобразования построено на решении системы линейных уравнений. Для того, чтобы система линейных уравнений имела единственное решение необходимо и достаточно, чтобы определитель матрицы этой системы был отличен от нуля. В данном случае, физический смысл этого критерия состоит в том, что все маркеры не должны лежать на одной прямой, а количество маркеров должно быть больше или равно трем.

4. Оценка сложности.

Решение задачи для каждого из объектов состоит из двух этапов:

1. определение коэффициентов системы линейных уравнений;
2. решение полученной системы линейных уравнений.
3. Первый этап имеет сложность $O(n_i)$, где n_i – количество маркеров, привязанных к i-му объекту. Вычисления на втором этапе занимают постоянное время. Таким образом, общая сложность вычислений для каждого объекта оценивается как $O(n_i) + const$.

Для получения координат произвольной точки объекта в мировой СК необходимо выполнить одну операцию умножения матрицы $4x4$ на вектор $4x1$.

Заключение

В данной работе рассмотрена проблема определения пространственного расположения 3D объектов сложной структуры. Задача была поставлена как определение положения произвольной

точки объекта в мировой системе координат при следующих основных требованиях к используемым алгоритмам: эффективность и точность. Необходимо было также учесть возможность работы в системе с ограниченными вычислительными ресурсами (бортовой комплекс ЭВМ).

Автором разработан подход для определения пространственного расположения объектов. Исходными данными является информация о координатах маркеров, закрепленных за объектами, а также априорная информация о сеточных моделях объектов. Предложенный автором алгоритм минимизации функции невязки определения координат основан на методе наименьших квадратов. Исследован критерий существования и единственности решения задачи и его физический смысл.

Ключевым мом для обеспечения эффективности предложенного автором подхода является использование априорной информации о поверхностях объектов совместно с использованием подхода "Motion Capture", что позволяет избежать усложнения вычислений при повышении сложности объектов.

Литература

1. Ф.Куафе. Взаимодействие робота с внешней средой. Москва, Мир, 1985.
2. Г.П.Катыс. Восприятие и анализ оптической информации автоматическими системами. Москва, Машиностроение, 1986.
3. Motion Capture overview
4. http://www.css.tayloru.edu/instrmat/graphics/hypgraph/animation/motion_capture/history1.htm
5. Motion capture system. http://www.polito.it/~bottino/MotionCapture/index.html
6. Ильин В.А., Ким Г.Д. Линейная алгебра и аналитическая геометрия. Издание Московского Университета, 1998.
7. Метод наименьших квадратов http://www.physics.spbstu.ru/physics/biblio/ExpData/2-3.html
8. Ильин В.А., Садовничий В.А., Сендов Бл.Х. Математический анализ. Издание Московского Университета, 1985.

Зленко П. А.

Интегрированная система паралингвистической обработки голоса[1]

Введение

Паралингвистика - неязыковые средства, используемые в речи, включающие в себя выражение в голосе индивидуального стиля говорящего (в том числе, качества голоса), эмоций, а также мимику и жестикуляцию. Рассмотрение паралингвистических свойств голоса с целью применения в программных системах началось сравнительно недавно, так что количество подобных разработок достаточно невелико - основное внимание исследователей сосредоточено на вопросах распознавания голоса. Тем не менее, использование паралингвистики при разработке некоторых типов систем, таких как автоматизированные синтезаторы речи, системы анализа звуковой сцены, автоматизированные системы речевого интерфейса способно существенно поднять качество сервиса. Кроме того, представляет определенный интерес создание эффект-процессоров, предназначенных для модификации и нормализации дикторской речи.

Настоящая работа посвящена рассмотрению возможного подхода к построению интегрированной системы, ориентированной на паралингвистический анализ и обработку речи. В первой части работы рассматриваются близкие по направлению концепции и системы, во второй части работы рассматривается предлагаемое направление разработок - интерактивная система обработки голоса.

Паралингвистические системы - краткий обзор

Данный обзор не претендует на всеобъемлющее покрытие аналогичных работ, однако должен дать достаточно ясную перспективу общих направлений деятельности.

В качестве двух основных направлений разработок в области паралингвистики следует назвать автоматический анализ голоса с целью обнаружения и классификации эмоционального состояния, а также

[1] При поддержке гранта РФФИ 02-07-90130

синтезаторы речи по тексту, имеющие возможность управления эмоциональной интонацией генерируемой речи.

Детекторы эмоций

Следует обратить внимание, что при реализации той или иной методики обнаружения и классификации эмоционального состояния необязательно рассматривать все зависимые от интонации параметры, более того, часто выгодно рассматривать именно выделенный подкласс параметров.

В работе [2] в рамках проекта VERBMOBIL рассматривается классификация фраз эмоционально/нейтрально при помощи нейронной сети. В качестве измеряемых характеристик голоса были использованы разнообразные просодические параметры - темп, характер пауз, громкость, длительность и F0 (фундаментальная частота). Кроме того, использовалась информация о части речи. Производилось как тестирование с использованием всех этих данных на уровне слов, так и на уровне фраз только по глобальным просодическим параметрам. Результаты для неиспользованных при обучении данных представлены в таблице (Таблица 1). Применялся набор аннотированных вручную реплик; в левом столбце представлены определенные оператором окраски, числовые данные даны по статистике ответов системы.

Таблица 1

	Глобальные параметры		Локальные параметры	
Класс	Попаданий	Точность	Попаданий	Точность
Нейтральный	61%	63%	79%	60%
Эмоциональный	67%	65%	52%	73%
В среднем	64%	64%	66%	67%

В другой работе тех же авторов ([3]) для набора из специально надиктованных с целью выражения неких эмоций фрагментов с использованием аналогичной методики было получено 86% попаданий.

В работе [1] для определения эмоциональной окраски использовалась статистика использования в речи так называемых интонационных шаблонов в соответствии со словарем IPO, составленным для датского языка (впрочем, похожие шаблоны можно применять и к другим языкам). Надиктованные дикторами фразы затем оценивались независимыми слушателями, которые определяли настрой

фрагмента. Определение шаблонов для фраз производилось вручную. Результаты свидетельствуют о применимости подобного подхода.

Система Feeltrace ([4]) позволяет определенным образом измерять эмоциональную окраску аудиовизуальных фрагментов. Feeltrace может работать в режимах только для аудио, только для видео и в комбинированном режиме. Результатом, выдаваемым Feeltrace, являются координаты на двумерной плоскости (активация, оценка), в определенном соответствии с [5]. Нетрудно видеть, что измерение "энергия" по Шлосбергу отсутствует. Следует заметить, что система измерений Шлосберга не годится для отображения таких эмоций, как любовь и беспокойство. Авторы Feeltrace не приводят результатов испытаний, которые достаточно трудно провести в данном случае, однако Feeltrace доступна для всех желающих.

Синтезаторы речи по тексту

При рассмотрении паралингвистики голоса можно выделить две основные составляющие: константные стиль и качество речи, и эмоциональную составляющую, которая определяет отклонения от стиля и качества. Стиль и качество речи включают в себя такие характеристики, как высота голоса, звонкость, акцент и т. п. Эмоциональная составляющая определяется вариациями просодических характеристик. В то же время, смысловая интонация речи, относящаяся к языковым средствам речи и привязанная к конкретному содержанию фразы, также передается при помощи аналогичных просодических вариаций. Тем не менее, данные направления изменений являются принципиально различными по назначению, поскольку эмоциональная составляющая не оказывает влияния на смысл сообщения. Перед разработчиками систем автоматического синтеза голоса, рассмотренных ниже, стояли, таким образом, две задачи по внесению в генерируемую речь интонационных вариаций.

Существует два основных класса синтезаторов, обладающих рассматриваемой функциональностью: формантные и конкатенативные, составляющие результат из данных банка фрагментов. Первые реализуют полностью искусственную генерацию в соответствии с некоторой моделью голосового тракта, наподобие популярного коммерческого синтезатора DECtalk, в синтезаторах второго типа обычно используется дифонный банк, дифоны (пары фонем) которого каким-то образом модифицируются и соединяются при синтезе.

К формантным синтезаторам, обладающим рассматриваемой функциональностью, относятся HAMLET ([6], [7]), Affect Editor ([8]) и

VAESS ([9]). HAMLET и Affect Editor используют в качестве конечного синтезатора DECtalk, VAESS - GLOVE.

Affect Editor позволяет вносить некоторое количество эмоциональных окрасок в синтезируемую речь, при этом требуемый эффект достигается путем установки комбинации из 17 глобальных параметров, в зависимости от желаемого эффекта. В экспериментах по опознанию эмоций людьми удалось добиться порядка 50% попаданий. Аналогично действует VAESS. Результаты испытаний ([9]) по опознанию слушателями эмоций для женского голоса, более сложного для полностью синтетической генерации, представлены в таблице (Таблица 2). Представлены количества соответствующих ответов слушателей, классифицирующих внесенные системой эмоциональные окраски.

Таблица 2

Отзывы слушателей	Представленная окраска			
	Ярость	Счастье	Грусть	Нейтрально
Ярость	9	3	1	1
Счастье	9	18	1	2
Грусть	4	0	20	12
Нейтрально	3	4	3	10

HAMLET позволяет управлять эмоциональной окраской путем задания координат в пространстве (активация, оценка, энергия).

В качестве примеров конкатенативных синтезаторов можно привести MARY ([10]) и VIECTOS ([12], [13]).

В системе VIECTOS используется банк фрагментов, содержащий LPC[1] коэффициенты и LPC остатки фрагментов. Применяется алгоритм SRELP[2], позволяющий менять F0 только за счет несложных преобразований LPC остатка (подробнее о SRELP см. [12]). Согласно [13], для формирования эмоциональной окраски используется управление F0, глобальное управление длительностями фонем, изменение громкости, наложение специальных линейных фильтров и некоторое управление качеством голоса. Результаты испытаний представлены в таблице (Таблица 3). Представлены количества соответствующих ответов слушателей, классифицирующих внесенные системой эмоциональные окраски.

---

[1] Кодирование с линейным предсказанием - сигнал представляется в виде коэффициентов линейного предсказания и остатка.

[2] Simple residual excited linear prediction

Таблица 3

Отзывы слушателей	Представленная окраска			
	Ярость	Грусть	Страх	Отвращение
Ярость	18	1	18	1
Грусть	1	31	1	24
Страх	9	9	8	10
Отвращение	17	4	18	10

В системе MARY используется в качестве конечного синтезатора MBROLA. Желаемая эмоциональная окраска задается в форме координат в пространстве (активация, оценка), наподобие Feeltrace. Соответствие акустических параметров и координат было определено путем автоматического анализа в SPSS (см. [11]).

Интерактивная система обработки голоса - концепция и задачи

При рассмотрениях вышеуказанных подходов заметны концептуальные ограничения, налагаемые классом системы. В случае детекторов рассматриваются, по сути, подмножества голосовых параметров, причем получаются, тем не менее, вполне достойные результаты. В случае генераторов голоса используется управление произвольным подмножеством параметров, при этом получается заметно сдвинутая в лучшую сторону статистика испытаний. Такие ограничения представляют, вообще говоря, существенный интерес, поскольку проявляются индивидуальные зависимости между параметрами и эмоциональной окраской. Однако, весьма отчетливо видно, что общая задача определения соответствия эмоций в речи не решается в полной мере, как хотелось бы. Результаты работы детекторов эмоций не дают достаточно высокого процента попаданий на произвольных данных, в то время как человек справляется с подобной задачей весьма успешно, а результаты испытаний синтезаторов речи на опознание слушателями эмоциональной окраски свидетельствуют, вообще говоря, лишь о похожести создаваемой окраски (то есть, слушатель может распознать), но не об абсолютной реалистичности эффекта. Можно утверждать, что методологическая неполнота, наблюдающаяся в двух обозначенных классах систем, обусловлена изолированностью задач.

В качестве одного из вариантов устранения данной методологической неполноты предлагается рассмотреть

(идеализированную) концепцию специфического речевого кодека[1]. Кодирование осуществляется на нескольких уровнях - на уровне постоянного стиля голоса конкретного человека (первый уровень), на уровне фонем (второй уровень) - кодер, фактически, осуществляет распознавание речи, - на уровне смысловой интонации (третий уровень) и на уровне эмоциональной интонации (четвертый уровень). Третий и четвертый уровни могут, в принципе, содержать достаточно быстро варьирующиеся данные, зависящие даже от некоторых случайных дефектов - к какому уровню они принадлежат и что значат, определяется, как нетрудно видеть, не диктором, чей голос кодируется, а слушателем и, соответственно, антропоморфным кодером. Если рассматривать задачу компрессии данного четырехуровневого представления, то основной выигрыш в размере данных должен возникать за счет сглаживания вариаций на третьем и четвертом уровнях и исключения случайных дефектов - информацию первого уровня можно вообще не включать в сообщения, а хранить отдельно в силу ее статичности. При декодировании вся четырехуровневая структура, естественно, превращается обратно в звуковой сигнал.

Если представить себе идеально соответствующую описанной выше концепции реализацию подобного кодека, то нетрудно видеть, что с помощью кодера решается проблема создания достоверного детектора эмоций - в качестве результата берутся данные четвертого уровня. Также решается проблема наложения эмоциональной окраски на синтезированную речь - на вход декодера подаются фиксированный стиль голоса, фонемы, смысловая интонация, сгенерированная неким алгоритмом, не являющимся частью кодека, и данные о требуемой окраске.

Такой кодек позволяет решать и другие, не менее интересные задачи, такие как высококачественная подмена индивидуальности голоса и высококачественная нормализация дикторской речи. Подмена индивидуальности голоса может осуществляться при помощи подстановки при декодировании чужого стиля на первом уровне. Нормализация дикторской речи может осуществляться путем сглаживания переходов на третьем и четвертом уровнях.

Реализация компрессии речи при помощи рассматриваемого кодека возможна, но не обладает особой практической ценностью, в частности поскольку, возможно, требуется определенная настройка под конкретный язык. Помимо этого, имеются также соображения целесообразности - существующие кодеки на базе LPC прекрасно выполняют свои задачи на медленных каналах связи, требуя, при этом, существенно меньшей вычислительной мощности. В конце концов,

[1] Кодер/декодер

устойчивость такого кодека к сильным помехам была бы неприемлемо низкой.

Практическая полная реализация данного кодека является затруднительной ввиду необходимости сохранять информацию о конкретной форме импульса, соответствующего фонеме. Нетрудно видеть, что сохранение такой информации на третьем и четвертом уровнях при необходимости обеспечить возможность восстановления без потерь потребует соответствующей информационной емкости, что неприемлемо. Поэтому предлагается рассматривать программную систему, в которой четырехуровневое представление используется в качестве способа аннотирования звуковой информации и средства задания некоторых преобразований. Как будет видно из дальнейшего рассмотрения задач подмены индивидуальности голоса и нормализации дикторской речи, такой подход является вполне целесообразным. Такая система должна также предоставлять возможность заменять разные варианты одной и той же фонемы или дифона, возможно, с выбором наилучшего варианта. В качестве способов применения вышеупомянутых преобразований к сигналу предлагается, в частности, использовать дифонный синтез в стиле SRELP, предоставляющий возможности по управлению F0, длительностями и формантными характеристиками.

Интерактивные возможности

Для решения задач генерации речи по тексту и анализа интонационной структуры фразы достаточно полностью автоматического режима работы декодера и кодера соответственно; для первой задачи такой режим является предпочтительным. Однако для решения задач подмены индивидуальности голоса и нормализации дикторской речи необходимы интерактивные возможности.

Подмена индивидуальности голоса

Задача высококачественной подмены индивидуальности голоса состоит в преобразовании надиктованной одним диктором речи к стилю голоса другого диктора. Стиль голоса второго диктора может быть получен из некоторой записи, возможно, небольшой длительности. Как нетрудно видеть, для решения такой задачи требуется не только подставить некоторые характерные для голоса второго диктора частотные и временные параметры (см., например, [14]), но, также,

возникает необходимость изменять более сложные характерные зависимости - управление ударениями в масштабе фраз, характерное использование эмоций и т. п. Вероятно, возможно построить систему, дающую возможность включить такие зависимости в стиль голоса, однако это представляется достаточно трудоемкой задачей. Кроме того, весьма возможно, что пользователь сочтет необходимым внести в результат преобразования некоторые изменения, придающие бо́льшую субъективную убедительность.

Нетрудно видеть, что для решения данной задачи требуется дать пользователю возможности по внесению изменений на втором уровне представления (для коррекции возможных языковых погрешностей), на третьем - для управления глобальными ударениями, например, - и на четвертом - для управления характерным использованием эмоций.

Нормализация дикторской речи

Задача нормализации дикторской речи состоит в обработке фонограммы с целью устранения разнообразных дефектов, как то: резких неестественных изменений интонации, неуместных оттенков эмоций, случайных неверных ударений и т. п. Безусловно, профессиональный диктор с большим опытом работы крайне редко допускает подобные погрешности, однако пользоваться услугами профессионала не всегда возможно и не всегда целесообразно.

При решении задачи нормализации существенную роль играет возможность ручного управления данными в четырехуровневом представлении. Нетрудно видеть, что, к примеру, затруднительно автоматически распознать неправильное ударение в масштабе фразы без использования лингвистического анализатора, а решить, где нужны какие оттенки эмоций, может, вероятно, только человек. Поэтому, при решении задачи нормализации дикторской речи требуется, как минимум, иметь возможность интерактивного вмешательства на третьем и четвертом уровнях представления.

Структура интерактивной среды

По-видимому, наиболее удобным отображением четырехуровневого представления в визуальной среде является размещение на рабочем столе трех графических образов, соответствующих второму, третьему и четвертому уровням,

параллельно на прокручиваемой по времени виртуальной ленте, в стиле большинства аудиоредакторов; просмотр данных первого уровня, стиля речи, должен быть выполнен в виде отдельного инструмента, интерфейс которого зависит от использованных алгоритмов на этом уровне. Данные второго уровня должны быть представлены в виде списка фонем, для которых можно изменять соответствие конкретным звукам в случае ошибок распознавания, имеется также возможность подбора дифона или фонемы. Данные третьего и четвертого уровней должны быть представлены в виде наборов интонационных измерений, в каждом из которых можно производить модификации. Должна иметься возможность оперативного прослушивания результата обратного преобразования в звук.

Архитектура предлагаемой системы

Архитектура предлагаемой системы должна предоставлять возможности по подключению произвольных реализуемых пользователем компонент анализа и обработки сигнала. В свете данного факта предлагается структура системы в форме модуля среды визуализации и управления (реализующего, в частности, управление алгоритмами визуализации данных первого уровня), модуля управления алгоритмами уровня стиля речи, модуля распознавания фонем (без возможности подключения пользовательских компонент, так как это не требуется), модуля управления алгоритмами уровня смысловой интонации, модуля управления алгоритмами уровня эмоциональной интонации и модуля управления алгоритмами анализа и обработки, которые используются алгоритмами всех четырех уровней. Также должен иметься модуль среды управления подключением алгоритмов.

Заключение

В работе рассмотрены некоторые вопросы построения интегрированных систем, ориентированных на паралингвистический анализ и обработку речи. Рассмотрены близкие по направлению концепции и системы, а также предлагаемое новое по сравнению с существующими направление разработок; произведены обоснование и оценка предложенного направления.

Литература

1. Mozziconacci, S.J.L., Hermes, D.J. 1999. Role of intonation patterns in conveying emotion in speech. Proc. ICPhS99, San Francisco, USA, стр. 2001-2004.

2. Huber, R. et al. 2000. Recognition of emotion in a realistic dialogue scenario. Proc. Int. Conf. on Spoken Language Processing, volume 1, Beijing, China, стр. 665-668.

3. Huber, R. et al. 1998. You BEEP machine - emotion in automatic speech understanding system. Proc. Workshop on TEXT, SPEECH and DIALOG (1998), стр. 223-228.

4. Cowie, R. et al. 2000. 'FEELTRACE': An instrument for recording perceived emotion in real time. Proc. ISCA Workshop on Speech and Emotion, 2000.

5. Schlosberg, H. 1954. Three dimensions of emotion. Psychol. Rev. 61 (2), стр. 81-88.

6. Murray, I.R., Arnott, J.L. 1995. Implementation and testing of a system for producing emotion-by-rule in synthetic speech. Speech Communication, 16, стр. 369-390.

7. Murray, I.R., Arnott, J.L. 1996. Synthesizing emotions in speech: is it time to get excited? Proc. of ICSLP 96 the 4th International Conference on Spoken Language Processing, Philadelphia, PA, USA, стр. 1816-1819.

8. Cahn, J.E. 1990. The generation of affect in synthesized speech. Journal of the American Voice I/O Society. Volume 8. стр. 1-19.

9. Bertenstam, J. et. al. 1997. The VAESS communicator: a portable communication aid with new voice types and emotions. Proc. Fonetik 97, the Swedish phonetics conference, Umea, стр. 57-60.

10. Schroeder, M., Trouvain, J. 2003. The German text-to-speech synthesis system MARY: a tool for research, development and teaching. International Journal of Speech Technology, 6, стр. 365-377.

11. Schroeder, M. et. al. 2001. Acoustic correlates of emotion dimensions in view of speech synthesis. Proc. Eurospeech 2001 vol. 1, стр. 87-90.

12. Rank, E., Pirker, H. 1998. VIECTOS - speech synthesizer, technical overview. Oesterreichisches Forschungsinstitut fuer Artificial Intelligence, Wien, TR-98-13.

13. Rank, E., Pirker, H. 1998. Generating emotional speech with a concatenative synthesizer. Proc. 5th International Conference on Spoken Language Processing (ICSLP'98), Sydney, Australia, стр. 671-674.

14. Галяшина, Е.И. Прикладные основы судебной фоноскопической экспертизы. Сб. Теория и практика судебной экспертизы, СПб, 2003.

Раздел III

Оптимизация вычислений

Саак А.Э.

Анализ функционирования многопроцессорных систем коллективного пользования

Введение

Вычислительные системы, мультипроцессорные устройства, компьютерные сети являются одним из самых востребованных ресурсов информатической цивилизации, которая всё более зримо проявляется в современном обществе [1, 2]. И тем большую актуальность приобретает анализ взаимодействия двух сред- поставляющей и потребляющей компьютерный сервис. Эффективность функционирования компьютерной системы зависит не только от её технологических характеристик, но и от параметров внешней по отношению к технической системе среды пользователей.

Переход от однопрограммного и однопользовательского режима к многопрограммному и многопользовательскому, от режима разделения времени к режиму разделения ресурса, от последовательных компьютеров к параллельным системам потребовал разработки моделей и методов, адекватно отражающих функционирование сложных вычислительных систем при обслуживании заявок пользователей.

Многопроцессорные системы ввиду их значительной компьютерной мощности и стоимости, достигающей десятков миллионов долларов, предполагают коллективное использование, обслуживание множества пользователей [3].

В работе определяются вероятностные характеристики многопроцессорной системы с множеством пользователей.

1. Постановка задач анализа функционирования многопроцессорных систем

Пусть имеется «k» независимых пользователей многопроцессорной системы с возможным количеством требуемых процессоров 1, 2, ..., N для каждого пользователя. Многопроцессорная система содержит M независимых процессоров. Требуется определить вероятностные характеристики многопроцессорной системы.

2. Определение воспринимательной способности многопроцессорной системы

Функционирование многопроцессорной системы с соблюдением правил неналожения процессоров, выделяемых разным пользователям, ограничения максимального индивидуального ресурса числом N и суммарного ресурса числом M принимается как комбинаторный эксперимент с подмножеством своих исходов, с указанием элементарных исходов, исчерпывающих в совокупности изучаемое множество.

Функционирование многопроцессорной системы состоит в удовлетворении заявок x_i на количество процессоров i- го пользователя, i = 1, 2, ..., k. Наиболее широкое множество целочисленных комбинаций $(x_1, x_2, ..., x_k)$ возникает при выполнении соотношения $\sum_{i=1}^{k} x_i \le M$, или близкой к нему системы неравенств

$$0 \le M - \sum_{i=1}^{k} x_i < N,$$

в которой учитывается только ограничение задачи на параметр M.

Симметризуя множество комбинаций x_i путём всевозможных перестановок номеров заявок, получаем кубическое множество объёмом M^k. Следовательно, мощность рассматриваемого множества равна $M^k/k!$, а мощностное отношение, или расширенная вероятность того же множества представляет собой величину

$$\frac{1}{k!}M^k \Big/ N^k = L^k/k!,\ L = [M/N],$$

при этом в качестве множества сравнения выбирается куб с ребром N.

Выделим в указанном множестве всевозможных заявок элементарные исходы системой неравенств

$$jN \le M - \sum_{i=1}^{k} x_i < (j+1)N,$$

в которой j = 0, 1, ..., L-1. Получим полновероятностную алгебру с расширенной вероятностью объемлющего исхода функционирования, приведённой выше.

Неравенства в левой части при j = 1 определяют ординарный однородный базис той же алгебры. В ординарных исходах нарушается правило функционирования многопроцессорной системы, связанное с параметром N. Вычитая объединение данного базиса исходов из объемлющего множества функционирования, придём к вероятности обслуживания не менее «k» пользователей, или пропускной

способности, многопроцессорной системы в виде формулы Лапласа [4, 5]

$$P(k,L)=\sum_{j=0}^{L-1}(-1)^j C_k^j (L-j)^k / k!,$$

в которой нулевое слагаемое было найдено выше, первое слагаемое имеет ту же структуру, что и нулевое, но отвечает кубическому множеству объёма $(M-N)^k$ с вероятностной мерой $\frac{1}{k!}(L-1)^k = \frac{1}{k!}(M-N)^k / N^k$ и т. д.

Данный вывод основан на предыдущем анализе комбинаций заявок пользователей ($x_1, x_2, \ldots, x_k$), в обслуживании которых состоит эксперимент функционирования многопроцессорной системы. При этом ограничение сверху на число «k» заявок пользователей не вводилось, что и приводит к коинтегральной форме найденного закона вероятности функционирования многопроцессорной системы.

Полученная полновероятностная алгебра позволяет выделить L элементарных исходов эксперимента функционирования многопроцессорной системы. Возможен элементарный исход, когда свободные пакеты процессоров отсутствуют, т.е. $M-\sum_{i=1}^{k} x_i < N$. Также возможна ситуация при которой имеется один свободный пакет процессоров $N \le M-\sum_{i=1}^{k} x_i < 2\cdot N$. Возможен элементарный исход при котором окажутся свободными два пакета процессоров $2\cdot N \le M-\sum_{i=1}^{k} x_i < 3\cdot N$ и т. д. вплоть до L–1 свободного пакета процессоров $(L-1)\cdot N \le M-\sum_{i=1}^{k} x_i < L\cdot N$. Ситуация при которой все L пакетов процессоров окажутся свободными невозможна, т.к. каждому пользователю, по условию задачи, требуется как минимум один процессор.

Абсолютные величины слагаемых формулы Лапласа $\sum_{j=0}^{L-1}(-1)^j C_k^j (L-j)^k / k!$ и представляют собой расширенные вероятности числа свободных пакетов процессоров.

Проведённые рассуждения позволяют рассмотреть целочисленное распределение

$$y_j(k,L) = C_k^j (L-j)^k / k!,\ j = 0, 1, \ldots, L\text{-}1,$$

называемое распределением Лапласа.

Теорема. При $L > k/2$ испытания Лапласа y_j обладают резко выраженным максимумом в виде одного или обоих концов некоторого целочисленного единичного отрезка.

Доказательство. Для наличия максимума требуется, чтобы $y_{j+1} / y_j = 1$, $j = 0, 1, \ldots, L\text{-}1$. Действительно,

$$\frac{y_{j+1}(k,L)}{y_j(k,L)} = \frac{C_k^{j+1}(L-(j+1))^k / k!}{C_k^j (L-j)^k / k!} = \frac{k-j}{j+1}\left(\frac{L-j-1}{L-j}\right)^k = 1 \text{ или}$$

$$\frac{k-j}{j+1} = \left(1 + \frac{1}{L-j-1}\right)^k.$$

Заметим, что функция, стоящая в левой части неравенства, является убывающей и при $j = 0$ имеет значение k; $j = L\text{-}1$ и условие $L > k/2$ приводят к значению функции не более 1. Функция, стоящая в правой части неравенства, является возрастающей и при $j = 0$ имеет значение порядка числа Эйлера $e \approx 2{,}718$. При $j = L\text{-}1$ значение функции $\to \infty$. Проведённый анализ показывает, что на интервале $j = 0, 1, \ldots, L\text{-}1$ при $L > k/2$, обе функции подпадают под признак Коши наличия корня, что и доказывает теорему.

Отметим, что наличие максимума у испытаний Лапласа $C_k^j (L-j)^k / k!$ на интервале $j = 0, 1, \ldots, L\text{-}1$ при $L > k/2$, аналогично свойству наличия максимума у испытаний Бернулли C_k^j на интервале $j = 0, 1, \ldots, k$.

После нормировки получаем закон распределения числа свободных пакетов процессоров в виде

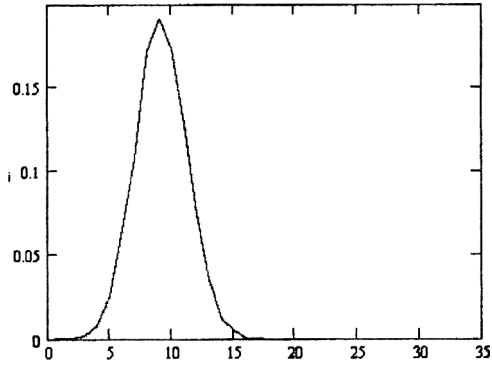
$$\frac{C_k^j (L-j)^k}{\sum_{i=0}^{L-1} C_k^i (L-i)^k},\ j = 0, 1, \ldots, L\text{-}1.$$

Таким образом, случайная величина j принимает значения $j = 0, 1, \ldots, L\text{-}1$ с вероятностями $C_k^j (L-j)^k \Big/ \sum_{i=0}^{L-1} C_k^i (L-i)^k$.

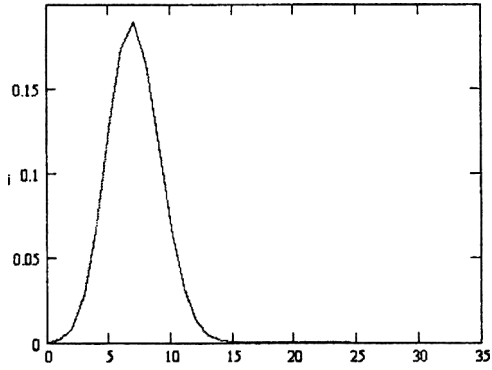
Отметим, что случайная величина (L–j), описывающая количество занятых пакетов процессоров, имеет тот же закон распределения.

Приведём результаты имитационного и аналитического моделирования закона распределения случайной величины j.

k=50, L=35



Имитационное моделирование



Аналитическое моделирование

Рис. Распределение свободных пакетов

Математическое ожидание величины свободных пакетов процессоров называем воспринимательной способностью многопроцессорной системы.

Воспринимательная способность характеризует возможности многопроцессорной системы по обслуживанию новых пользователей.

Воспринимательная способность многопроцессорной системы с параметрами k, L равна величине

$$m_k(L) = \sum_{j=1}^{L-1} jC_k^j (L-j)^k \Big/ \sum_{i=0}^{L-1} C_k^i (L-i)^k .$$

Результаты расчёта приведены на графике.

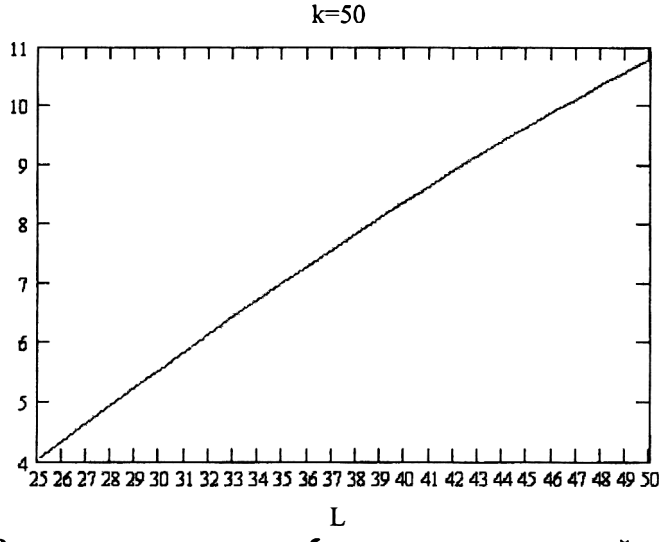
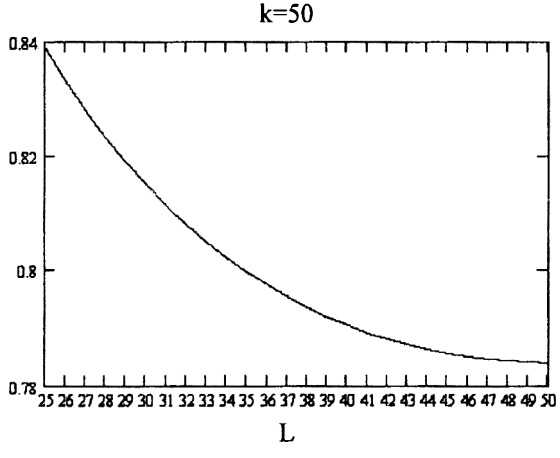


Рис. Воспринимательная способность многопроцессорной системы

Определим коэффициент загрузки многопроцессорной системы как отношение математического ожидания числа занятых пакетов процессоров к общему числу пакетов в системе. При этом получаем следующие графики зависимости коэффициента загрузки от L и k.



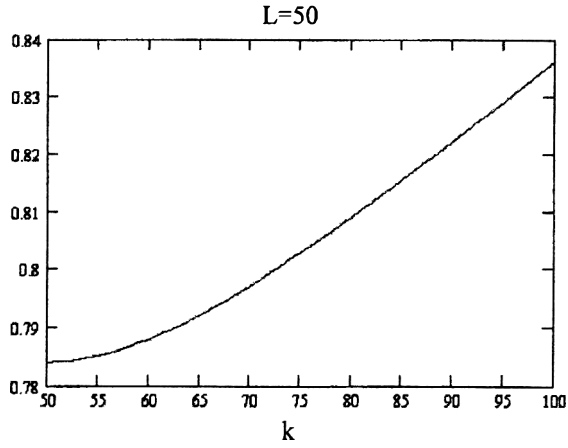


Рис. Коэффициент загрузки многопроцессорной системы

Видим, что показатель загрузки многопроцессорной системы находится в диапазоне 75%-85% на интервале $L \in (k/2,k)$.

3. Закон распределения вероятностей числа одновременно обслуживаемых пользователей

В разделе 2 была указана вероятность обслуживания многопроцессорной системой не менее «k» пользователей. Найдём закон распределения вероятностей числа одновременно обслуживаемых пользователей. Для этого из вероятности числа не менее «k» одновременно обслуживаемых пользователей

$$\frac{1}{k!}\sum_{j=0}^{L-1}(-1)^j C_k^j (L-j)^k$$

надо вычесть вероятность числа не менее «k+1» одновременно обслуживаемых пользователей

$$\frac{1}{(k+1)!}\sum_{j=0}^{L-1}(-1)^j C_{k+1}^j (L-j)^{k+1},$$

при этом получим

$$\frac{1}{k!}\sum_{j=0}^{L-1}(-1)^j C_k^j (L-j)^k - \frac{1}{(k+1)!}\sum_{j=0}^{L-1}(-1)^j C_{k+1}^j (L-j)^{k+1} =$$

$$= \frac{1}{k!}\sum_{j=0}^{L-1}(-1)^j C_k^j (L-j)^k \left(1 - \frac{L-j}{k+1-j}\right).$$

Найдём математическое ожидание числа одновременно обслуживаемых пользователей.

$$m(k)=\sum_{k=L}^{M}\frac{k}{k!}\sum_{j=0}^{L-1}(-1)^j C_k^j (L-j)^k\left(1-\frac{L-j}{k+1-j}\right)$$

Результаты численного расчёта приведены в таблице.

L	4	5	6	7	8	9
m(k)	7.667	9.667	11.667	13.667	15.667	17.667

Видим, что с точностью до целых результаты расчёта совпадают со значением величины 2L, которая качественно представляется средним значением числа одновременно обслуживаемых пользователей. Действительно, каждый пользователь требует в среднем N/2 процессоров. Следовательно, одновременно можно обслужить 2L пользователей.

Аналогично находим дисперсию D(k) и среднеквадратическое отклонение σ(k) числа одновременно обслуживаемых пользователей. Результаты расчётов приведены в таблице.

L	4	5	6	7	8	9
D(k)	2.889	3.556	4.222	4.889	5.556	6.222
σ(k)	1.7	1.886	2.055	2.211	2.357	2.494

Видим, что с точностью до целых чисел результаты вычисления σ(k) совпадают со значением величины $\sqrt{L}$.

Заметим, что m(k), D(k), σ(k) зависят не от абсолютных значений величин M и N, а от их отношения, что согласуется с качественными представлениями о функционировании многопроцессорной системы.

Заключение

В работе изучена случайная величина «j», отвечающая среде, поставляющей компьютерный сервис, на основе которой определяется воспринимательная способность многопроцессорной системы, характеризующая возможности по обслуживанию новых пользователей. Определён коэффициент загрузки многопроцессорной системы как отношение математического ожидания числа занятых пакетов процессоров к общему числу пакетов в системе. Описывается случайная величина «k», характеризующая среду, потребляющую компьютерный сервис. Совокупность данных случайных величин позволяет

значительно полнее представить функционирование многопроцессорной системы с множеством пользователей.

Литература

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ–Петербург, 2002.– 608с.
2. Методы и средства обработки информации: Труды первой Всероссийской научной конференции/ Под ред. Л.Н. Королева. – М.: Изд–во МГУ, 2003. – 580 с.
3. Немнюгин С.А., Стесик О.Л. Параллельное программирование для многопроцессорных вычислительных систем. – СПб.: БХВ–Петербург, 2002. – 400с.
4. Макаревич О.Б., Саак Э.М., Чефранов А.Г. Анализ загруженности однородных микропроцессорных вычислительных систем коллективного пользования // Автоматика и вычислительная техника, 1980, №4, с. 32-36.
5. Сачков В.Н. Вероятностные методы в комбинаторном анализе. М.: Наука, 1978. – 288с.

Веселов Н. А.

Алгоритм расчёта средней задержки пакета в сетях передачи данных типа wormhole[1]

Введение

Настоящая работа посвящена исследованию сетей передачи данных типа wormhole [10]. Данный класс сетей (Myrinet, Servernet II, Sunfinity) применяется при построении кластерных вычислительных комплексов, а также специализированных систем реального времени.

Основной характеристикой любой сети передачи данных является скорость её работы, то есть насколько быстро информация может быть доставлена от отправителя до получателя по сети. При использовании пакетной передачи, данная величина представляет собой задержку пакета в сети, обусловленную следующими двумя составляющими.

1. *Ёмкостью каналов* сети, или скоростью передачи каждого отдельного канала. Это физическая величина, которая является фиксированной для любой конкретной сети, её увеличение представляет собой задачу разработки сетевого оборудования и нами не рассматривается.
2. *Временем*, которое пакет проводит в *ожидании в промежуточных узлах* и которое обусловлено текущим состоянием сети: множеством абонентов, используемыми ими маршрутами передачи пакетов и создаваемой нагрузкой на сеть.

Задержка пакета есть случайная величина, поэтому целесообразно рассматривать среднюю задержку пакета, которая для любой конкретной сети с фиксированной ёмкостью каналов является обобщённой характеристикой и в конечном итоге определяет пропускную способность системы [1, 2].

Сети передачи данных типа wormhole

Основной особенностью рассматриваемого нами класса сетей является используемая в них технология передачи данных. Информация передаётся в виде пакетов, каждый из которых, образно говоря, можно представить в виде червя, прогрызающего себе путь от узла –

[1] Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект № 02-01-00261

отправителя до узла – получателя через промежуточные коммутаторы. При этом, «голова червя» может быть уже принята в узле – получателе, в то время как хвост ещё находится в узле – отправителе. Пакет как бы растягивается по сети, таким образом, что в любой промежуточной точке (промежуточном коммутаторе) находится только небольшая его часть [4]. Для реализации описанной концепции, каждый пакет разбивается на небольшие (возможно размером 1 байт), неделимые единицы информации, не имеющие логической структуры – флиты (flit). Флит являются минимальной частью пакета, которая может храниться в промежуточном коммутаторе [10].

В сетях типа wormhole возможно возникновение ситуации *блокировки*, когда несколько пакетов пытаются проложить соединение через один и тот же канал сети. В этом случае заблокированные пакеты не теряются, а ожидают освобождения указанного канала сети, при этом они не освобождают уже занятых каналов, что может повлечь за собой блокировку других пакетов и т.д.. При освобождении канала среди ждущих пакетов выбирается следующий для передачи в соответствии с т.н. *правилом выбора входного канала*. Далее в настоящей работе мы будем рассматривать сети, использующие *круговое правило выбора* (round-robin), при котором следующий входной канал для передачи выбирается по кругу. В сетях без промежуточной буферизации пакетов теоретически возможно возникновение *тупиковых ситуаций* взаимной блокировки нескольких пакетов, при которых дальнейшее движение заблокированных пакетов по сети не может возобновиться. Большинство существующих сетей являются безтупиковыми, что обеспечивается соответствующим выбором путей передачи пакетов. В настоящей работе также будут рассматриваться безтупиковые сети.

Формальная постановка задачи

Мы воспользуемся традиционным подходом к постановке задачи, принятым в теории сетей массового обслуживания, и формально определим:

1. структуру сети и используемую технологию передачи данных;
2. нагрузку на сеть;
3. маршрутизацию в сети.

Структура сети и используемая технология передачи данных

Сеть задаётся при помощи ориентированного графа $G = (U, V)$, где U - вершины (узлы сети), V - дуги (каналы связи). Множество вершин представляет собой объединение двух непересекающихся подмножеств $U = U_1 \cup U_2, U_1 \cap U_2 = \varnothing$, подмножества оконечных устройств U_1 и подмножества коммутаторов U_2. Будем полагать, что каждый коммутатор является *полносвязным* (т.е. способен соединить любой входной канал с любым выходным каналом, не занятым передачей), у любого оконечного устройства имеется *единственный канал* подключения к сети, *топология сети фиксирована*. Будем считать, что для каждого отдельного канала известна скорость передачи или ёмкость, одинаковая для всех каналов сети.

Опираясь на обзор [10], суть технологии передачи wormhole можно сформулировать в виде следующих принципов.

1. Для передачи каждого отдельного пакета между отправителем и получателем устанавливается *монопольное соединение* – последовательность каналов сети, передающих флиты *только данного пакета*.
2. В процессе установки соединения *возможно возникновение блокировки*, при которой пакет приостанавливает своё движение, но не теряется и не освобождает уже занятых им каналов сети.
3. Если имеется несколько заблокированных пакетов, ожидающих освобождения одного и того же канала, то используется *круговое правило выбора* среди ждущих пакетов. Для всех коммутаторов определён параметр *время переключения* (обозначим его d), представляющий собой задержку выходного канала коммутатора на переход к следующему входному каналу при круговом просмотре.
4. После установки соединения передача пакета представляет собой *непосредственное копирование данных из памяти узла – отправителя в память узла – получателя*. Скорость передачи при этом равна скорости работы отдельного канала сети.
5. При разрыве соединения *каналы сети освобождаются последовательно по направлению от отправителя к получателю*.

6. При установке соединения и при его разрыве *отсутствуют задержки на распространение сигнала* в каналах сети. Например, если в сети имеется единственная пара абонентов, то при прохождении заголовка пакета по каналам происходящем при установке соединения, возникающие задержки связаны только с перекоммутацией каналов в промежуточных коммутаторах и обусловлены параметром *d*, введённым выше.

Нагрузка на сеть

Нагрузка, поступающая в сеть определяется двумя составляющими: *множеством абонентов сети* и *генерируемым ими трафиком*. Абоненты сети представлены множеством тяготеющих пар отправитель – получатель $W = \{w = (j,k), j \in U_1, k \in U_1, j \neq k\}$, причём и отправитель и получатель являются оконечными устройствами. Трафик (поток пакетов), генерируемый каждой парой абонентов $w = (j,k)$, определяется двумя случайными величинами A^w и B^w. Случайная величина A^w представляет собой время между двумя последовательными поступлениями пакетов в узел отправитель *j*, а B^w - время, которое уходит на передачу данных пакета после установки соединения между отправителем *j* и получателем *k* и которое зависит от длины пакета и ёмкости отдельных каналов сети. Способ задания указанных случайных величин существенно опирается на методы, используемые при построении алгоритма расчёта, и будет подробно рассмотрен далее в настоящей работе.

Будем считать, что выполняется ряд предположений, которые являются традиционными для проводимого нами анализа сетей передачи данных.

1. *Ёмкость буферной памяти у оконечных устройств является достаточно большой*, чтобы память не переполнялась при работе сети. Данное предположение определяет то, что в сеть не поступает избыточного трафика, который последняя не способна принять полностью.

2. *Множество абонентов и создаваемая ими нагрузка фиксированы* и не изменяется в процессе функционирования сети. Последнее предположение обусловлено тем, что изменение множества абонентов и создаваемой ими нагрузки происходит относительно редко по сравнению с задержкой пакета и связано с высокой скоростью работы каналов связи.

Мы будем рассматривать стационарную работу системы, при которой сеть работает в устойчивом режиме, а характеристики случайных процессов, протекающих в системе, не изменяются со временем.

Маршрутизация в сети

Будем полагать, что в сети используется *статическая однопутевая маршрутизация*, то есть для любой пары абонентов $w = (j,k)$ фиксирован единственный путь $p_w = (j, i_1(j,k), i_2(j,k), ..., i_{m_w}(j,k), k)$, который не меняется при функционировании сети. Все промежуточные узлы, образующие путь, являются коммутаторами, а узлы – абоненты – оконечными устройствами.

Будем считать, что пути передачи выбраны таким образом, что в сети *невозможно возникновение тупиковых ситуаций* взаимной блокировки абонентов.

Постановка задачи

Для формального определения задержки пакета в сети введём несколько дополнительных обозначений.

Рассмотрим любую пару абонентов $w = (j,k)$, канал l, используемый данной данной парой абонентов и коммутатор $i \in U_1$, для которого канал l является выходным каналом. Обозначим через T_l^w случайную величину – время, которое уходит на то, чтобы пакету данной пары абонентов занять канал l после того как заголовок пакета попал в коммутатор i, а через L^w - случайную величину – время, которое уходит у пакета данной пары абонентов $w = (j,k)$ на то чтобы установить соединение между отправителем j и получателем k и передать данные пакета после установки соединения.

Рассмотрим любое оконечное устройство сети $j \in U_1$, которое является отправителем в нескольких парах абонентов. Обозначим через Q^j случайную величину – время ожидания начала установки соединения для пакета, поступившего в узел j, в очереди этого узла.

Средние задержи пакета в сети представляют собой математические ожидания случайных величин T_l^w, L^w и Q^j, для обозначения которых мы будем использовать символы $E[T_l^w]$, $E[L^w]$ и $E[Q^j]$ соответственно.

Постановка задачи, формулируется следующим образом. Необходимо разработать алгоритм, который по заданным графу сети $G = (U, V)$, множеству абонентов $W = \{w = (j,k), j \in U_1, k \in U_1, j \neq k\}$, создаваемой каждой парой абонентов нагрузке (случайные величины A^w и B^w) и используемым маршрутам передачи $p_w = (j, i_1(j,k), i_2(j,k), ..., i_{m_w}(j,k), k)$, позволит рассчитывать среднюю задержку пакета при занятии определённых каналов сети $E[T_l^w]$, среднюю задержку на установку соединения между отправителем и получателем и передачу данных пакета $E[L^w]$, и, для любого оконечного устройства $j \in U_1$, которое является отправителем, среднюю задержку в очереди ждущих пакетов в данном узле $E[Q^j]$.

Алгоритм расчёта задержек в сети

Предлагаемый в настоящей работе алгоритм опирается на широко известный метод теории сетей массового обслуживания [1, 2], суть которого состоит в том, что сеть разбивается на отдельные каналы, после чего производится анализ работы каждого канала как независимой системы. Все действия при расчёте средних задержек в сети условно можно разбить на три этапа.

1. Определение последовательности, в которой будет производиться анализ отдельных каналов сети. Первый шаг призван определить зависимости между задержками при занятии отдельных каналов сети и учесть эти зависимости в ходе работы алгоритма.
2. Анализ отдельных каналов сети как систем массового обслуживания. На втором шаге каждый канал рассматривается отдельно от остальных каналов после чего, с помощью методов теории массового обслуживания, производится расчёт задержки при его занятии. Результаты затем используются при анализе последующих каналов сети в порядке, определяемом п. 1.
3. Расчёт задержки пакетов на установку соединения и передачу данных для каждой пары абонентов и задержки в очереди ждущих

пакетов для каждого оконечного устройства - отправителя производится с использованием результатов работы предыдущего шага алгоритма.

Далее мы рассмотрим указанные шаги более подробно.

Разбиение сети на отдельные каналы и определение последовательности проведения расчётов

Необходимость рассмотрения отдельных каналов сети в определённом порядке может быть проиллюстрирована следующим простым примером.

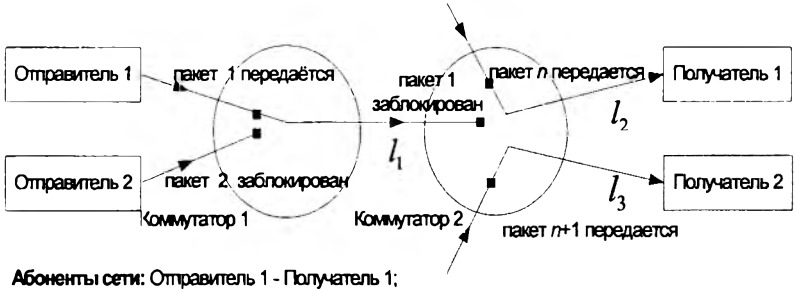


Рис. 1. Пример зависимости задержки при занятии каналов сети от задержек при занятии других каналов сети.

На рис. 1 изображён фрагмент сети, включающий две пары абонентов (два узла отправителя и два узла получателя). Так как эти пары абонентов используют один и тот же канал сети l_1, то блокировка любого из них на одном из последующих каналов (l_2 или l_3) приводит к увеличению времени, которое тратится другим абонентом на занятие канала l_1. Действительно, пакет 1 (рис. 1) будучи заблокированным и ожидая занятия канала l_2 не освобождает уже занятого им канала l_1, что приводит к увеличению времени ожидания занятия канала l_1 пакетом 2. Таким образом, для определения задержки пакета любой пары абонентов при занятии канала l_1, необходимо сначала определить задержки, возникающие при занятии каналов l_2 и l_3, то есть решить ту же самую задачу для каналов l_2 и l_3.

В работе [9] доказано следующее

Утверждение. Для сетей типа wormhole, в которых *невозможно возникновение тупиковых ситуаций*, существует последовательность рассмотрения каналов сети, позволяющая рассчитать среднюю задержку пакета при занятии каждого канала.

Опираясь на это утверждение, определим последовательность рассмотрения отдельных каналов сети следующим образом. Занумеруем все каналы $l_1, l_2, ..., l_n$, все пары абонентов $w_1, w_2, ..., w_m$ и составим таблицу маршрутов, в которой перечислены все каналы, используемые абонентами сети (рис. 2). Канал сети, который является *j*-м по счёту в маршруте *i*-й пары абонентов будем обозначать $l(i, j)$. Необходимо отметить, что любой канал сети может использоваться несколькими различными парами абонентов, поэтому некоторые элементы в таблице могут повторяться, например, $l(i_1, j_1) = l(i_2, j_2) = l_k$ для некоторых i_1, i_2, j_1, j_2, k.

$$\begin{pmatrix} l(1,1), l(1,2), ..., l(1,k_1) \\ ... \\ l(m,1), l(m,2), ..., l(m,k_n) \end{pmatrix}$$

Рис. 2. Множество маршрутов, используемых абонентами сети для передачи пакетов.

Рассмотрим некоторую пару абонентов w_i, которой соответствует *i*-я строка в таблице маршрутов, и некоторый *j*-й по счёту канал в этом маршруте $l(i, j)$. Для упрощения обозначений вместо записи $T_{l(i,k)}^{w_i}$ будем использовать запись $T_{i,k}$. Время, на которое канал $l(i, j)$ занимается пакетом *i*-й пары абонентов, представляет собой случайную величину (обозначим её $B_{i,j}$). Случайная величина $B_{i,j}$ равна сумме $B_{i,j} = \sum_{k>j} T_{i,k} + B^{w_i}$. Предположим, что канал $l(i, j)$ используется ещё в некоторой паре абонентов, например w_p на *q*-м месте, то есть $l(i, j) = l(p, q) = l$. Тогда очевидно, что для определения задержки $T_{p,q}$, которая возникает при занятии канала l пакетом пары абонентов w_p, необходимо знать величину $B_{i,j}$, а для определения задержки $T_{i,j}$ - величину $B_{p,q}$. Из определения

случайных величин $B_{i,j}$ и $B_{p,q}$ следует, что они зависят от $T_{i,k}, k > j$ и $T_{p,r}, r > q$. Таким образом, для определения $T_{i,j}$ и $T_{p,q}$ необходимо сначала определить $T_{i,k}, k > j$ и $T_{p,r}, r > q$. Данное требование верно для всех абонентов сети, использующих канал *l*.

Заполним *список каналов* в порядке, определяющем последовательность их рассмотрения. Первоначально данный список пуст. Для пары абонентов w_i будем просматривать таблицу маршрутов с правой стороны (от получателя к отправителю), начиная с элемента $l(i, k_i)$. Каждый просматриваемый элемент таблицы соответствует некоторому каналу сети. Данный элемент будет помечаться в том случае, если для любой пары абонентов, использующей указанный канал, все элементы таблицы маршрутов, стоящией правей его, уже помечены. Пометка будет ставиться на всех элементах таблицы, соответствующих данному каналу сети. Так, элемент таблицы $l(i, k_i)$ и все равные ему элементы, могут быть сразу же помечены, потому что соответствующий канал сети идёт непосредственно к оконечному устройству и может встречаться в любом маршруте только на последнем месте. Помеченный канал будет помещаться на последнее место в *список каналов*.

Предположим, что на некотором шаге алгоритм пытается пометить элемент таблицы маршрутов $l(i, j) = l$. Если для любой пары абонентов w_p, использующей канал *l* на некотором *q*-м месте ($l(i, j) = l(p, q) = l$) все элементы таблицы маршрутов $l(p, r), r > q$ уже помечены, то все элементы таблицы маршрутов, соответствующие каналу *l*, также могут быть помечены. Если для некоторой пары абонентов w_p такой, что $l(i, j) = l(p, q) = l$, существует непомеченный канал $l(p, r), r > q$, стоящий правее рассматриваемого канала *l* в *p*-й строке таблицы маршрутов (соответствующей паре абонентов w_p), то необходимо выбрать самый правый непомеченный элемент в этой строке и попытаться пометить его. Таким образом, происходит переход с *i*-й строки таблицы на *p*-ю.

Если в результате нескольких переходов между строками таблицы маршрутов мы снова вернёмся на *i*-ю строку, то в сети возможно возникновение тупиковой ситуации взаимной блокировки абонентов, что иллюстрируется примером рис. 3 для трёх пар абонентов w_1, w_2 и w_m. Если указанные абоненты одновременно начнут

передачу, причём пакет пары w_1 займёт канал 3, пакет пары w_2 займёт канал 1 и пакет пары w_m займёт канал 2, то в сети возникнет тупиковая ситуация.

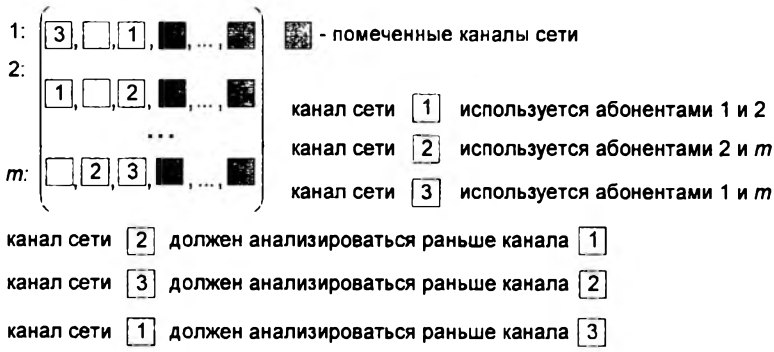


Рис. 3. Пример сети, в которой возможно возникновение тупиковой ситуации.

Так как нами рассматриваются безтупиковые сети, то на каждом шаге алгоритму удастся найти следующий канал для пометки. По окончанию работы первого шага будет получен список всех каналов сети в порядке их просмотра при проведении расчёта задержек.

Необходимо сделать следующее замечание. Выберем любую пару абонентов сети $w_i = (j,k),\ j,k \in U_1$ и рассмотрим первый канал $l(i,1)$ в маршруте, используемом данной парой абонентов. В таблице маршрутов указанный канал может встречаться в некоторых других строках, отличных от *i*-й. Пусть, например, он также используется некоторой парой абонентов $w_p, (p \neq i)$. Тогда для *p*-й пары абонентов отправителем также является узел *j*, так как в качестве промежуточных узлов при движении пакета могут выступать только коммутаторы, а рассматриваемый нами канал сети выходит из оконечного устройства *j*. Следовательно, рассматриваемый канал может стоять в *p*-й строке таблицы маршрутов также только на первом месте, то есть $l(p,1) = l(i,1)$. Таким образом, любой канал, который является первым в пути некоторой пары абонентов, также является первым в маршруте любой другой пары абонентов сети, использующей его.

Применение метода анализа во множестве дискретных моментов времени для расчёта отдельного канала сети

Для анализа работы отдельного канала сети нами будет использоваться система массового обслуживания с круговым опросом и N очередями (рис. 4). Каждой очереди соответствует некоторый входной канал коммутатора, по которому поступают пакеты для передачи по рассматриваемому выходному каналу. Выходной канал производит круговое сканирование входных каналов и, если в некоторой очереди имеются ждущие передачи пакеты, то передаёт пакет из этой очереди. После окончания передачи входной канал переходит к просмотру следующей очереди по кругу, затратив некоторое время на переключение. Пакет удаляется из очереди и освобождает занятое им место также только после окончания передачи.

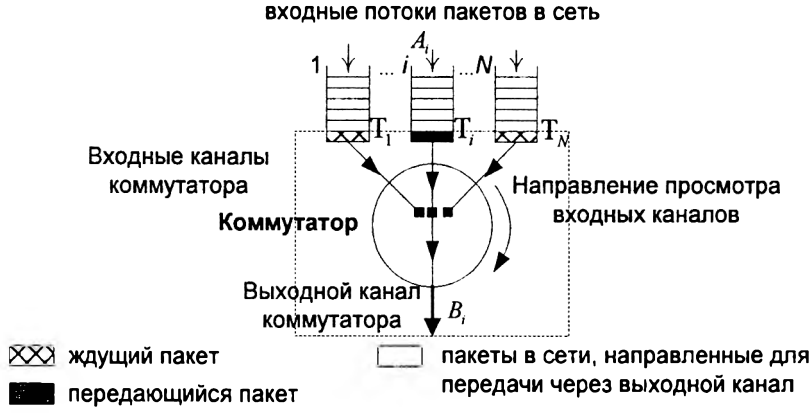


Рис. 4. Система для представления работы отдельного канала сети.

Очереди в системе на рис. 4 формируются из пакетов, которые уже поступили в сеть и должны проследовать через рассматриваемый нами канал сети, но пока не дошли до этого канала. Пакеты, находящиеся на первых местах в каждой очереди, уже проложили соединение до анализируемого канала и ждут момента занятия этого канала. Передающийся пакет – это пакет, который уже занял рассматриваемый нами канал сети и либо прокладывает соединение через последующие каналы маршрута, либо передаёт данные от отправителя к получателю через установленное соединение.

Поток пакетов, поступающий в i-ю очередь, определяется случайными величинами A_i и B_i, где случайная величина A_i задаёт

время между двумя последовательными поступлениями пакетов, а случайная величина B_i определяет время, на которое пакет *i*-го входного потока занимает выходной канал. Время переключения между входными каналами *i* и *(i+1)* mod *N* задаётся случайной величиной S_i. Необходимо определить время, которое пакет каждого потока проведёт на первом месте в очереди, то есть задержку с того момента, как будет проложено соединение к рассматриваемому коммутатору до момента, когда пакет займёт выходной канал коммутатора. Данное время является случайной величиной, которую мы будем обозначать через T_i для *i*-й очереди. Анализ отдельного канала сети сводится к определению распределения случайных величин T_i для всех входных потоков 1, 2, ..., *N*.

Для решения поставленной задачи в алгоритме расчёта используется метод анализа во множестве дискретных моментов времени (discrete-time domain analysis), предложенный P. Tran-Gia (1986 г.). Полное описание данного метода можно найти в работах [6, 8, 12], мы же лишь кратко изложим его суть.

Метод анализа во множестве дискретных моментов времени является численным и позволяет посредством нескольких итераций найти интересующие нас случайные величины. Ось времени разбивается на небольшие интервалы одинаковой фиксированной длины, после чего определяются дискретные моменты времени, соответствующие выбранным интервалам. Применимость данного подхода обусловлена тем, что в реальной системе все события также происходят в определённые дискретные моменты времени, связанные, например, с тактовой частотой работы процессора.

Так как система рассматривается в стационарном (устойчивом) режиме работы и любое изменение её состояния может происходить только в отмеченные моменты времени, то для описания системы достаточно использовать дискретные случайные величины. Дискретная случайная величина *X* определяется при помощи дискретного распределения вероятностей $x(k), k \in Z$, где $x(k) = P(X = k)$. Таким образом, в рассматриваемом случае (рис. 4), должны быть заданы распределение времени $a_i(k)$ между последовательным поступлениями пакетов в *i*-ю очередь (с.в. A_i), распределения времени $b_i(k)$ занятия выходного канала системы пакетом *i*-го потока (с.в. B_i) и распределение времени $s_i(k)$ переключения между входными каналами *i* и *(i+1)* mod *N* (с.в. S_i). Необходимо определить

распределение $\tau_i(k)$ времени ожидания занятия выходного канала пакетом, поступившим по *i*-му входному каналу (с.в. T_i).

Метод анализа во множестве дискретных моментов времени позволяет рассчитать распределения $\tau_i(k)$ для всех входных потоков и тем самым решает поставленную задачу. Единственное дополнительное предположение, которое используется нами при применении указанного метода, состоит в том, что любой *i*-й поток пакетов является простейшим. Указанное предположение будет обсуждаться в следующем пункте настоящей работы.

Проведение расчёта средней задержки пакетов для всей сети

Прежде, чем рассмотреть последний шаг работы алгоритма, отметим, что случайные величины A^w и B^w, определяющие входной поток пакетов в сеть, являются дискретными, поэтому мы будем предполагать, что трафик, поступающий в сеть, задаётся распределениями $a^w(k)$ и $b^w(k)$.

Расчёт задержек, возникающих при занятии отдельных каналов сети, ведётся по таблице маршрутов (рис. 3), а каналы сети последовательно выбираются из *списка каналов*, сформированного в начале работы алгоритма. После анализа каждого канала он исключается из списка, а все элементы таблицы маршрутов, соответствующие данному каналу, помечаются. Первоначально все элементы таблицы маршрутов являются непомеченными. Для каждой строки таблицы *i* введём случайную величину B_i с распределением $b_i(k)$, которая представляет собой сумму задержки, возникающей при установке соединения через уже просмотренные и помеченные каналы пакетом пары абонентов w_i, и задержки на передачу данных пакета после установки соединения. Перед началом проведения расчётов распределение данной случайной величины инициализируется распределением времени передачи пакетов, которое задано при определении нагрузки на сеть, то есть $b_i(k) = b^{w_i}(k)$. Способ построения списка просмотра каналов на первом шаге алгоритма гарантирует, что при рассмотрении любого канала *l*, для любого

элемента $l(i,j)$ таблицы маршрутов, соответствующего этому каналу ($l(i,j)=l$), все каналы, стоящие правее $l(i,j)$ в i-й строке таблицы, будут уже просмотрены и помечены. Таким образом, случайная величина B_i представляет собой время, на которое канал l занимается пакетом i-й пары абонентов.

Пусть на некотором шаге рассматривается канал l, который является выходным для коммутатора $y \in U_2$. В соответствие с замечанием, сделанным при определении первого шага алгоритма, для указанного канала сети возможны два случая. Либо для некоторой пары абонентов канал l является не первым каналом соответствующего маршрута, либо указанный канал находится на первом месте в маршруте некоторой пары абонентов, тогда ему могут соответствовать только элементы таблицы маршрутов, занимающие самые левые позиции в соответствующих строках.

Рассмотрим *первый вариант* расположения канала l. В этом случае, для любого элемента $l(i,j)$ таблицы маршрутов, соответствующего каналу l ($l(i,j)=l$) определён предыдущий канал маршрута, то есть существует элемент $l(i,j-1)$. Обозначим все входные каналы коммутатора y, по которым в выходной канал l могут поступить пакеты, через $l_0, l_1, ..., l_{r-1}$ (всего r каналов), причём увеличение индекса соответствует направлению сканирования, то есть сначала просматривается канал l_0, потом каналы l_1, $l_2, ..., l_{r-1}$ в порядке следования, затем снова канал l_0 и т.д. по кругу.

Для применения метода анализа во множестве дискретных моментов времени, множество абонентов сети, использующих канал l, разобьём на классы в соответствии с номером входного канала коммутатора y, который используется для передачи. Таким образом, будет сформировано множество классов $W_{l_p,l} = \{w_i : \exists j, l(i,j)=l, l(i,j-1)=l_p\}$, p=0,1,...,r-1. Некоторые из указанных классов могут быть пустыми, если по соответствующему входному каналу коммутатора не поступает пакетов для передачу по каналу l.

Каждый класс $W_{l_p,l}$ формирует поток, идущий по каналу l_p сети и направленный на канал l. Сделаем следующее

Предположение 1. Для всех абонентов из множества $W_{l_p,l}$ задержки на занятие канала l одинаковы, то есть $T_l^{w_i} = T_l^{w_j}, \forall w_i, w_j \in W_{l_p,l}$.

Существуют примеры ситуаций, когда данное предположение не выполняется. Однако, как показывают результаты экспериментов, при достаточно большом количестве различных узлов сети, участвующих в передаче, указанное предположение может использоваться при проведении расчётов. Необходимо отметить, что подобное предположение используется также в работах [3, 9].

Для каждого множества $W_{l_p,l}$ определим суммарную интенсивность $\lambda(W_{l_p,l})$ поступления пакетов от абонентов данного класса по формуле $\lambda(W_{l_p,l}) = \sum_{w \in W_{l_p,l}} \frac{1}{E[A^w]}$ и сделаем следующее

Предположение 2. Поток пакетов, создаваемый множеством абонентов $W_{l_p,l}$ является простейшим с параметром $\lambda(W_{l_p,l})$.

Данное предположение обусловлено тем, что поток пакетов, который должен быть передан по определённому каналу сети, формируется под влиянием достаточно большого количества абонентов. Это, как известно из теории массового обслуживания, приводит к формированию простейшего потока. Необходимо отметить, что аналогичное предположение делается в работах [5, 9], посвящённых сетям класса wormhole, а также в классической работе [2].

Используя величины $\lambda(W_{l_p,l})$ и функцию плотности экспоненциального распределения, для каждого входного канала l_p такого, что $W_{l_p,l} \neq \varnothing$, построим распределение $a_p(k)$ времени между двумя последовательными поступлениями пакетов для передачи через каналы l_p и l.

Далее, для абонентов класса $W_{l_p,l}$, усреднением величин B_i по интенсивностям отдельных потоков, вычислим распределение времени $t_p(k)$, на которое пакеты абонентов данного класса занимают канал l, то есть $t_p(k) = \sum_{w_i \in W_{l_p,l}} \frac{b_i(k)}{\lambda(W_{l_p,l}) E[A^{w_i}]}$. Будем использовать следующее

Предположение 3. Случайные величины B_i для потоков $w_i \in W_{l_p,l}$ являются независимыми.

Условия выполнения данного предположения аналогичны условиям выполнения *предположения 1*.

Наконец, определим задержки при переключении между соседними входными каналами, ведущими передачу по каналу *l*. Пусть l_p и l_q два входных канала коммутатора *y* таких, что $W_{l_p,l}, W_{l_q,l} \neq \varnothing$, причём, по всем входным каналам, расположенным между l_p и l_q по направлению сканирования, передачи не ведётся, то есть $W_{l_c,l} = \varnothing, c \in \{(p+1) \bmod r, (p+2) \bmod r, ..., (q-1) \bmod r\}$. Тогда задержка на переключение между каналами *p* и *q* равна $((q-p+r) \bmod r) \cdot d$, где *d* – задержка на переключение между соседними каналами коммутатора, определённая в п. 0 настоящей работы, а распределение $s_p(k)$ имеет вид

$$s_p(k) = \begin{cases} 1, k = ((q-p+r) \bmod r) \cdot d \\ 0, k \neq ((q-p+r) \bmod r) \cdot d \end{cases}.$$

Используя распределения $a_p(k)$, $t_p(k)$, $s_p(k)$ и метод анализа, описанный в предыдущем пункте, произведём расчёт распределений случайных величин задержек $\tau_p(k)$ при занятии канала *l* для всех классов $W_{l_p,l}$.

Для всех пар абонентов w_i, использующих канал *l*, произведём пересчёт случайных величин задержек B_i по формуле $B_i + \mathrm{T}_{i,j}$. Для этого сделаем следующее

Предположение 4. Задержки, с которыми пакет сталкивается при занятии определённых каналов сети, не зависят от будущих задержек при занятии следующих каналов сети по маршруту следования пакета.

Условия выполнения данного предположения аналогичны условиям выполнения *предположения 1* и *предположения 3*.

Опираясь на *предположение 4*, для всех абонентов сети w_i, использующих канал *l* для передачи, произведём пересчёт распределений $b_i(k)$ по формуле $(b_i \oplus \tau_i)(k)$, где

$(u \oplus v)(k) = \sum_{i+j=k} u(i) \cdot v(j)$ - свёртка дискретных распределений $u(k)$ и $v(k)$. После проведения расчётов, все элементы таблицы, соответствующие каналу *l*, помечаются, а канал *l* удаляется из списка просматриваемых каналов сети.

Рассмотрим *второй вариант* расположения канала *l*. В этом случае $L^{w_i} = B_l$, так как для любой пары абонентов сети w_i, использующей канал *l*, он является первым в маршруте передачи пакетов.

К концу просмотра списка каналов, нами будут вычислены распределения случайных величин T_l^w и L^w (обозначим их $\tau_l^w(k)$ и $l^w(k)$ соответственно), что даёт возможность определить средние задержки $E[T_l^w]$ и $E[L^w]$.

На последнем шаге расчётов необходимо вычислить среднее время ожидания начала отправки в очередях оконечных устройств сети. Рассмотрим некоторое оконечное устройство $j \in U_1$ и все пары абонентов, для которых данное устройство является отправителем. Обозначим это множество $W_j = \{w = (j,k), k \in U_1, w \in W\}$. Для всех потоков пакетов, поступающих в узел *j*, известны распределение времени между двумя последовательным поступлениями и распределение времени на установку соединения и отправку данных ($a^w(k)$ и $l^w(k)$, $w \in W_j$ соответственно). Обозначим через $a_j(k)$ и $l_j(k)$ соответствующие характеристики *суммарного потока пакетов* , поступающего в узел *j*.

Очевидно, что время ожидания в очереди узла *j* определяется, во-первых, характеристиками поступающего в данный узел суммарного потока пакетов, а во-вторых, принципами, используемыми при организации очереди и выборе следующего пакета для отправки среди ждущих пакетов. Таким образом, любой конкретный случай поступающей нагрузки и организации работы оконечных устройств требует отдельного рассмотрения. Для проведения экспериментальной проверки мы воспользуемся предположением о независимости потоков пакетов, поступающих в узел *j*, и представим распределения характеристик суммарного потока, поступающего в данный узел, при помощи усреднения распределений $a^w(k)$ и $l^w(k)$ по интенсивностям

потоков отдельных пар абонентов, то есть $a_j(k) = \sum_{w \in W_j} \frac{a^w(k)}{\lambda(W_j) E[A^w]}$ и $l_j(k) = \sum_{w \in W_j} \frac{b^w(k)}{\lambda(W_j) E[B^w]}$, где $\lambda(W_j) = \sum_{w \in W_j} \frac{1}{E[A^w]}$ - интенсивность суммарного потока пакетов. Для расчёта распределения времени ожидания Q^j и среднего $E[Q^j]$ также будем использоваться метод анализа во множестве дискретных моментов времени по схеме, изложенной в работе [11].

Пример работы алгоритма расчёта

Работа метода расчёта будет проиллюстрирована на следующем наглядном примере. Рассмотрим фрагмент сети, построенной с использованием топологии Ч. Клоза, применяемой на практике при построении реальных сетей [7]. Рассматриваемый фрагмент состоит из 8 оконечных устройств и 5 коммутаторов (рис. 5).

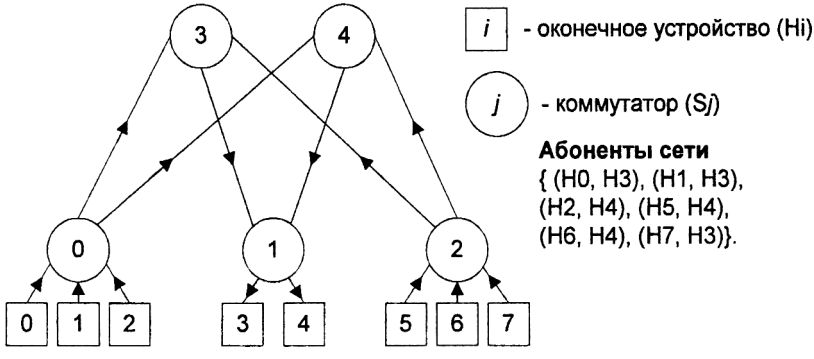


Рис. 5. Фрагмент сети передачи данных. Пример работы алгоритма расчёта.

Каждая пара абонентов создаёт одинаковую нагрузку на сеть со следующими характеристиками: A^w - дискретная аппроксимация простейшего потока с интенсивностью 0.05 $пакетов/\mu s$, B^w - дискретная аппроксимация экспоненциального распределения с параметром 0.33 $1/\mu s$, задержка на переключение $d = 0.01 \mu s$.

Ниже приведены таблицы для нескольких абонентов и каналов сети, позволяющие сравнить результаты работы алгоритма расчёта с результатами имитационного моделирования.

Канал сети	Входные каналы	Средняя задержка при занятии канала $E[T_i^*]$ (μs)		
		Моделирование	Расчёт	Погрешность
Комм. 1 – ок. ус-во 4.	0 1	0.527 0.935	0.563 0.963	7% 3%
Комм. 3 – комм. 1.	0 2	0.170 0.174	0.165 0.165	3% 5%
Комм. 0 – комм. 3.	3 4	0.169 0.217	0.151 0.222	11% 2%

Абоненты сети	Средняя задержка на установку соединения и передачу данных $E[L^*]$ (μs)		
	Моделирование	Расчёт	Погрешность
Ок. Ус-во 0 – ок. ус-во 3	7.15	7.51	5%
Ок. Ус-во 7 – ок. ус-во 3	4.93	5.38	9%

Оконечные устройства	Средняя задержка в очереди ждущих отправки пакетов $E[Q_j]$ (μs)		
	Моделирование	Расчёт	Погрешность
Ок. ус-во 0	4.51	4.70	4%
Ок. ус-во 7	1.95	2.10	8%

Как видно из приведённых таблиц, погрешность работы алгоритма расчёта лежит в пределах 10% - 15%, что позволяет говорить о его применимости для решения поставленной задачи.

Заключение

В настоящей работе предложен алгоритм расчёта важнейшей характеристики функционирования сетей передачи данных типа

wormhole – средней задержки пакета в сети. Указанный алгоритм опирается на методы теории массового обслуживания и сетей массового обслуживания. В частности, при его построении был использован метод анализа во множестве дискретных моментов времени, предложенный P. Tran-Gia (1986 г.). Проведённая серия имитационных экспериментов продемонстрировала точность предлагаемого метода.

Литература

1. Д. Бертсекас, Р. Галлагер. Сети передачи данных. – М: Мир, 1989.
2. Л. Клейнрок. Коммуникационные сети. Стохастические потоки и задержки сообщений. – М: Наука, 1970.
3. С.Я. Шоргин. Модели сетей связи со смешанной нагрузкой. Техника средств связи, Серия СС, 1985 г., вып. 1, с. 60 – 63.
4. N. Boden, D. Cohen, R. Felderman, Al Kulawic, C. Seitz, J. Seizovic, and W. Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, vol 15, no 1, 1995.
5. W.J. Dally. Performance analysis of *k*-ary *n*-cube interconnection networks. *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 775-785, 1990.
6. R. Dittmann, F. Hüebner. Discrete-Time Analysis of a Cyclic Service System with Gated Limited Service. *Research Report Series No. 67*, Institute of Computer Science, University of Würzburg, June 1993, http://www-info3.informatik.uni-wuerzburg.de/TR/tr067.pdf.
7. Guide to Myrinet 2000 Switches and Switch Networks, 2001, http://www.myri.com/myrinet/m3switch/guide.
8. D.-S. Lee, B. Sengupta. An Approximate Analysis of a Cyclic Service Queue with Limited Service and Reservations. *Queueing Systems*, 11, 1992, pp. 153 – 178.
9. O. Lysne. Towards an Analytical Model of Wormhole Routing Networks. *Microprocessors and Microsystems*, vol.21, pp. 491-498, 1998.
10. L.M. Ni, P.K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, pp. 62-76, 1993.
11. O. Rose. Interdeparture Time Correlations of the Discrete-time GI/GI/1 Queue. *Research Report Series No. 204*, Institute of Computer Science, University of Würzburg, May 1998, http://www-info3.informatik.uni-wuerzburg.de/TR/tr204.pdf.
12. P. Tran-Gia. Analysis of Polling Systems with General Input Process and Finite Capacity, *IEEE Transactions on Communications*, vol. 40, no. 2, 1992, pp.337-344.

Кичигин Д.Ю.

Методы планирования потока инструкций для конвейерных RISC архитектур[1]

Введение

Планирование инструкций (instruction scheduling) – это семейство программных и аппаратных техник, направленных на повышение скорости выполнения потока команд за счет параллельного выполнения отдельных машинных операций.

Впервые такие техники стали применяться с появлением машин CRAY-1 и CDC 6600 в 1960гг. Наличие в этих машинах возможности совмещения операций[2] обусловило потребность в эффективных алгоритмах планирования, управляющих порядком выполнения инструкций так, чтобы, с одной стороны, не была нарушена логика выполнения программы и в тоже время максимально использовалась возможность совмещения операций.

Существует два вида техник планирования – это динамические (или аппаратные) техники (управление потоком инструкций осуществляется аппаратурой компьютера и происходит динамически во время выполнения программы) и статические (программные) техники, когда планирование инструкций осуществляется статически на одной из стадий компиляции программы.

Наибольшее развитие идеи статического планирования получили в конце 70гг – 80гг. Одной из причин этого было появление VLIW машин. Характерной особенностью таких машин является

---

[1] Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект № 02-01-00261

[2] "совмещение операций" – метод проектирования архитектуры компьютера, при котором аппаратура компьютера в любой момент времени выполняет одновременно более одной базовой операции. Включает в себя два понятия: параллелизм (совмещение операций достигается путем воспроизведения в нескольких копиях аппаратной структуры) и конвейеризацию (разделение подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры) [1].

"длинный" формат команды (*Very Large Instruction Word*) – когда в одну команду компилятором помещается несколько инструкций, и которые затем одновременно выдаются на выполнение[1]. В этих системах основная нагрузка по оптимизации потока инструкций ложилась на компилятор, что и послужило толчком к более сильному развитию программных средств планирования.

90-е гг. охарактеризовались большим коммерческим успехом RISC- и CISC-архитектур. Это породило необходимость адаптации к ним существующих методик планирования, т.к. эти архитектуры обладали меньшими ресурсами для обеспечения совмещения операций (появилась проблема нехватки регистров), и как следствие, техники планирования стали более специализированными[2].

Настоящая статья посвящена обзору программных(статических) техник планирования инструкций. В главе "Общий обзор алгоритмов" мы приведем обзор основных существующих техник программного планирования потока команд. В главе "Алгоритмы планирования для конвейерных RISC-архитектур" мы рассмотрим планирование инструкций в контексте конвейерных RISC-архитектур.

Общий обзор алгоритмов

Алгоритмы планирования могут быть классифицированы по следующим критериям[2]:

Область применения алгоритма

Различают локальные алгоритмы планирования (*local scheduling*), глобальные алгоритмы(*global acyclic scheduling*) и циклические алгоритмы (*cyclic scheduling*).

Локальные алгоритмы работают с линейными блоками(*basic blocks*)[1]. Областью применения глобальных алгоритмов являются несколько расположенных вместе линейных блоков без циклов.

---

[1] Линейным блоком называется последовательность команд не содержащая переходов

Циклические алгоритмы применимы к нескольким линейным блокам где могут встречаться переходы назад(циклы).

Архитектура компьютера, для которого предназначается алгоритм

Здесь важны следующие параметры:

Конечные/неограниченные ресурсы

Каждая инструкция выполняется за единичную задержку или имеются инструкции, выполнение которых занимает более одного такта

Каждая инструкция имеет простой шаблон использования ресурсов (т.е. каждая инструкция на каждой стадии конвейера использует только один ресурс), либо существуют инструкции с более сложными шаблонами

Нужно сказать, что реальные компьютеры имеют конечные ресурсы, в них присутствует несколько операций с задержкой более чем один такт и также встречаются команды с комплексными шаблонами использования ресурсов.

Природа алгоритма

Здесь алгоритмы варьируются от простых однопроходных, которые расставляют каждую инструкцию раз и навсегда, до алгоритмов, выполняют многопроходный и достаточно сложный поиск расписания. Также существуют итеративные алгоритмы, которые строят последовательность элементарных изменений к существующему расписанию чтобы приблизить его к итоговому расписанию.

Алгоритмы локального планирования

Задачей алгоритма локального планирования является построение оптимального расписания инструкций внутри одного линейного блока. Т.к. известно что задача построения такого расписания является NP-полной, то основу решения этой задачи составляет нахождение адекватных эвристик[3]. В алгоритме локального планирования сочетаются невысокая потребность в

вычислительных ресурсах (это однопроходный алгоритм) и построение расписания, в большинстве случаев достаточно близкого к оптимальному. Кроме того алгоритм не требует значительных изменений при работе с инструкциями, выполняющимися более чем с одной задержкой.

Одним из направлений развития алгоритма является развитие модели архитектуры для которой он предназначен. Начиная от начальной модели, где каждая инструкция имеет простой шаблон ресурсов (на каждом цикле используется только один ресурс) и единичную задержку, были построены обобщения на инструкции с комплексными шаблонами использования ресурсов и с произвольными задержками. Итогом стало использование полностью обобщенного «микрошаблона» (*microtemplate*) использования ресурсов, разработанного Токоро[2].

Алгоритмы глобального планирования

Одним из недостатков алгоритма локального планирования является область его применения. В среднем каждый линейных блок имеет длину 5-20 инструкций [2], что не позволяет достичь большого выигрыша в производительности, т.к. возможность переупорядочить инструкции не нарушив при этом логики выполнения программы сильно ограничена. Для решения этой проблемы были предложены глобальные подходы, смысл которых заключается в одновременном планировании содержимого сразу нескольких линейных блоков.

Планирование расширенного линейного блока

Это одна из самых простых техник глобального планирования. Под расширенными линейным блоком (EBB – Extended Basic Block) подразумевается такая последовательность линейных блоков B_1 ,..., B_k, где любой B_i, $1 \le i < k$ является единственным предшественником B_{i+1}, а B_1 может иметь несколько предшественников. Планирование EBB аналогично локальному планированию плюс добавляются дополнительные ограничения на перемещение операций между переходами.

Планирование трасс

Одной из первых техник глобального планирования является предложенное Фишером (Fisher)[2] планирование трасс. Суть этого подхода заключается в построении трассы линейных блоков – последовательности линейных блоков выполняющихся друг за другом, и дальнейшем планировании содержимого трассы как единого линейного блока. При построении трасс учитывается частота выполнения их в программе, т.е. в планировании участвуют наиболее часто используемые пути (основываясь на информации из профайла)[3].

Планирование суперблоков

Эта техника является разновидностью планирования трасс. Суперблоки похожи на трассы за исключением того, что в процессе построения некоторые блоки дублируются с целью удаления общих точек входа. Сначала строятся трассы, затем производится дублирование хвостов с целью удаления входов в трассу со стороны. В процессе дублирования дублируются все блоки от первой точки входа со стороны до конца трассы. Затем все входы 'извне' в трассу перенаправляются в получившуюся копию. Дублирование затем повторятся уже на полученной копии. Поэтому суперблок может иметь несколько точек перехода, но одну точку входа.

Планирование гиперблоков

Еще одним решением в области глобального планирования является планирование гиперблоков. Эта техника имеет много общего с планированием суперблоков. Здесь к коду применяется процедура IF-Conversion[4], которая устраняет все внутренние ветвления. В результате получается код, под названием гиперблок(*hyperblock*), похожий на суперблок, который тоже имеет одну точку входа и несколько выходов. Планирование гиперблока осуществляется аналогично суперблоку, за исключением того что любые две операции с разными предикатами (т.е. операции, встречающиеся в разных ветках программы – которые не могут присутствовать на одной трассе выполнения программы) могут быть распланированы так, чтобы использовать одни и те же ресурсы. После этого к коду применяется обратная процедура Reverse IF-Conversion.

Циклическое планирование

До сих пор мы рассматривали алгоритмы, применимые к коду не содержащему переходов назад(или циклов). Такие алгоритмы принято называть ациклическими (*acyclic*). Алгоритмы, работающие с кодом содержащим переходы назад относят к циклическим(*cyclic*) алгоритмам.

Самой простой техникой циклического планирования, является техника разворачивания циклов (*loop unrolling*). Суть этого подхода состоит в разворачивании тела цикла на несколько итераций и применения к полученной последовательности линейных блоков, не содержащих переходов назад, одной из техник глобального планирования [2][5].

Другим направлением циклического планирования является группа методов, объединенных под общим названием Программная конвейеризация (*software pipelining*). Идея этих методов была позаимствована у разработчиков конвейерных устройств и состоит в частичном перекрытии (наложении друг на друга) выполнения соседних итераций цикла – каждая новая итерация начинает выполняться до окончания предыдущей.

Одним из методов программной конвейеризации является техника модульного планирования (*modulo scheduling*). Этот метод строит одно и тоже расписание для каждой итерации цикла и каждую очередную итерацию запускает на выполнение с некоторой постоянной скоростью – через так называемый интервал инициализации (*initiation interval*) – ИИ. Это условие известно как Ограничение Модульного планирования (*modulo scheduling constraint*) и гарантирует что каждый ресурс будет использоваться каждой итерацией не больше чем один раз за каждый промежуток времени, равный ИИ[2][5]. Для модульного планирования был установлен минимальный интервал инициализации (МИИ), т.е. были выведены ограничения на максимальную скорость с которой могут выполняться итерации цикла[2][5].

Изначально алгоритм модульного планирования работал только с простейшими циклами, телом которых является простой линейных блок, каждая операция имеет простой шаблон использования ресурсов и т.д. В настоящее время известно множество модификаций этого алгоритма, способные работать с более сложным кодом (снимаются ограничения на простоту инструкций, допускаются более сложные структуры кода внутри итераций). Появились итерационные методы

модульного планирования – сначала строится расписание с ИИ равным минимальному ИИ, в случае неудачи ИИ увеличивается, и так до тех пор, пока не построим расписание.

Алгоритмы планирования для конвейерных RISC архитектур

Отличительной особенностью RISC-архитектур (по сравнению VLIW-машинами) является небольшое наличие параллелизма в командах что, как следствие, не дает планировщику возможности достичь большого прогресса в скорости выполнения результирующего кода. Кроме того, небольшое количество регистров делает непривлекательными техники агрессивного планирования[2]. Все это приводит к тому что планирование инструкций для RISC-машин рассматривается как более второстепенный процесс (в контрасте с VLIW-машинами, где процесс планирования инструкций занимает центральное место).

Но в тоже время развитие рынка RISC-машин и их коммерческий успех породили потребность в эффективных методах планирования инструкций, разработанных именно для них.

Планирование и распределение регистров

В RISC-архитектурах, вследствие ограниченного числа регистров, становится актуальной известная задача[7] распределения регистров(*register allocation*).

Поэтому основной акцент исследований в области планирования инструкций для RISC-машин касается вопроса организации друг относительно друга процессов распределения регистров и собственно планирования инструкций[2][6].

Существует несколько подходов к решению этой проблемы.

Первая группа методов состоит из техник, где фаза распределения регистров предшествует планированию инструкций. Основным аргументом в пользу этих методов является то, что хотя код с лучшим расписанием предпочтительней, но если он требует большего количества регистров чем доступно – он просто не будет работать. Поэтому достижение нужного распределения регистров более важно чем построение хорошего расписания.

Другая группа состоит из техник где, наоборот, планирование предшествует распределению регистров. Здесь планировщик имеет большую свободу действий и, соответственно, больше возможностей

для построения лучшего расписания. Недостатком таких методов является то, что более короткие расписания имеют тенденцию увеличивать регистровое давления. Кроме того, в случае если построенное расписание требует большее количество регистров чем доступно, то необходимо добавлять код для запоминания значений регистров (*register spill code*). Это будет делать построенное расписание менее эффективным, т.к. планирование (построение расписания) происходило без учета кода запоминания регистров и потери от этого кода могут превышать преимущества построенного расписания.

В некотором смысле компромиссным подходом являются алгоритмы, где процесс планирования выполняется два раза – один раз до распределения регистров (*pre-scheduling*) и один раз после (*post-scheduling*). На первой фазе выполняется планирование с учетом на некоторое ограничение регистрового давления, потом выполняется распределение регистров. Здесь, при необходимости, добавляется код для сохранения значений регистров. Затем, на последней фазе, происходит финальное планирование с учетом результатов первого планирования.

Такой подход позволяет построить расписание более приближенное к оптимальному. К недостаткам этого метода можно отнести то что все решения принимаемые на каждой фазе принимаются без учета их последствий для последующей фазы.

И, наконец, последним подходом, позволяющим уйти от недостатков предыдущих алгоритмов, является одновременное планирование и распределение регистров. Идея состоит в том чтобы совместить процессы планирования и распределения регистров: каждый раз когда происходит планирование очередной инструкции для ее результата выбирается свободный регистр. Этот регистр освобождается, если его больше не использует ни одна инструкция. В случае если нет свободных регистров, выбирается один из используемых и сохраняется его значение.

Заключение

В данной статье был проведен обзор алгоритмов для решения задачи статического(программного) планирования инструкций. Были рассмотрены алгоритмы локального, глобального и циклического планирования. Также были исследованы особенности разработки алгоритмов для конвейерных RISC архитектур.

Литература

1. В.З.Шнитман, С.Д.Кузнецов, Аппаратно-программные платформы корпоративных информационных систем. Информационно-аналитические материалы Центра Информационных Технологий, МГУ. - М.: 1998.

2. B.Ramakrishna Rau, Joseph A.Fisher, Instruction-Level Parallel Processing: History, Overview and Perspective. J. Supercomputing, Vol. 7, No.1/2, 1993, pp. 9-50.

3. Д. В. Калашников, И.В. Машечкин, М.И. Петровский, Планирование потока инструкций для конвейерных RISC архитектур. Вестник МГУ №4 1999 (с. 39-44)

4. Майкл С. Шланскер, Б. Рамакришна Рау, Явный параллелизм на уровне команд, Открытые системы, #11-12/1999 16.11.1999.

5. Alexandre E. Eichenberger, Modulo Scheduling, Machine Representations, and Register-Sensitive Algorithms. PhD thesis, University of Michigan, Department of Electrical Engineering and Computer Science, Ann Arbor, MI, 1996.

6. David A.Berson, Rajiv Gupta, Mary Lou Soffa, Integrated Instruction Scheduling and Register Allocation Techniques. Languages and Compilers for Parallel Computing, LCPC'98, LNCS 1656, pp.247-262, 1999.

7. А.Ахо, Р.Сети,Д.Ульман, Компиляторы: принципы, технологии и инструменты. : Пер. с англ. – М.: Издательский дом «Вильямс»,2001.-768с.

Корухова Л. С., Малышко В. В.

Ассоциативные и дедуктивные методы в планировании многошаговых многовариантных задач[1]

Введение

Настоящая работа выполнена в рамках исследования подходов к автоматическому решению сложных задач на ЭВМ и преследует цель – создание программных средств для их реализации. При этом сложными задачами считаем задачи, в ходе решения которых возникают несколько возможных путей или вариантов и полный перебор таких вариантов часто оказывается неэффективным или практически неосуществимым. В качестве примеров подобных задач можно привести:

1. шахматные задачи,
2. геометрические задачи на вычисление величин и/или доказательство свойств,
3. задачи планирования целенаправленных действий, например, перемещений робота и т. п.

Рассматриваемый в работе подход (см. [Корухова и Любимский, 1978]) предполагает построение систем автоматического решения на основе методов планирования адекватных тем принципам, которыми руководствуется человек при поиске решений и составлении плана целенаправленных действий.

При этом мы считаем, что главный из этих принципов – направленный поиск достаточно хорошего решения или плана действий. Направление поиска должно определяться на основе конкретной задачи и конкретной обстановки в ходе планирования. Действительно, в реальных задачах, во-первых, требуется найти достаточно хороший план достижения цели, хотя и не обязательно оптимальный, и, во-вторых, перебор в таких задачах просто невозможен, поэтому поиск плана, естественно, должен быть целенаправленным. В процессе планирования, учитывая особенности построения плана человеком, необходимо также рассматривать и сравнивать несколько альтернативных планов, однократно уточнять

[1] Работа выполнена при поддержке РФФИ (№02-01-000281).

одинаковые участки альтернативных планов, использовать в процессе планирования различные стратегии и приемы планирования. Отметим, что в 1984 году известный американский ученый Дуглас Б. Ленат в статье "Искусственный интеллект" высказал созвучное мнение о подходе к решению сложных задач: "Ключ к решению задач искусственного интеллекта лежит в сокращении перебора вариантов при поиске решения. Для этого программы должны реализовать те же принципы, которыми в процессе мышления пользуется человек" [Ленат, 1984].

Эти принципы положены в основу решателя геометрических задач (РГЗ) – модельной системы, в рамках которой проводится исследование методов решения задач и реализация программных средств планирования. Предметной областью РГЗ являются планиметрические задачи на вычисление величин геометрических объектов и/или доказательство их свойств. Решатель реализован средствами языка Пролог (среда программирования Arity Prolog32 1.1). При работе над этой системой удалось разработать и реализовать различные методы планирования и комбинирование различных методов при решении задач. Далее в первой части мы рассматриваем структуру РГЗ, а во второй части обсуждаем реализованные методы и стратегии планирования.

1. Структура РГЗ

Построение плана в РГЗ осуществляется *программой-администратором* с привлечением *базы знаний*. Администратору задается формулировка задачи, включающая цель (нахождение искомой величины или доказательство факта), а результатом его работы является план решения задачи, представляющий собой последовательность шагов по достижению поставленной цели. В процессе планирования администратор управляет выбором и применением стереотипов, хранящихся в базе знаний решателя. Описание стереотипа состоит из *ситуации*, определяющей условия, в которых стереотип применим, и *действий*, которые выполняются при применении стереотипа. Стереотипы также имеют характеристики, на основании которых администратор осуществляет их отбор.

В ходе планирования решения производится построение *графа планирования решения (ГПР)*, и осуществляется вывод новых фактов из условий задачи (в зависимости от стратегии планирования). Узлы ГПР соответствуют подцелям, а воображаемые дуги – переходам от одной подцели к другой (см. Рис.1а). Цели в графе уточняются в

результате применения стереотипов, при этом с уточненной целью связывается одна или несколько цепочек подцелей. План считается завершенным, когда в ГПР появляется путь из начальной вершины в концевую, состоящий из достигнутых целей, которым не требуется дальнейшее уточнение (см. Рис.1б), или когда в ходе вывода добавляется факт, соответствующий поставленной в задаче цели.

Именно граф планирования является в решателе той структурой, которая поддерживает планирование в соответствии с перечисленными выше принципами. В отличие от известных систем планирования, которые исходят из того, что реализуют один из вариантов перебора ветвей дерева логического вывода, с использованием графа появляется возможность уточнять однократно общие участки планов и рассматривать несколько альтернативных планов в ходе планирования.

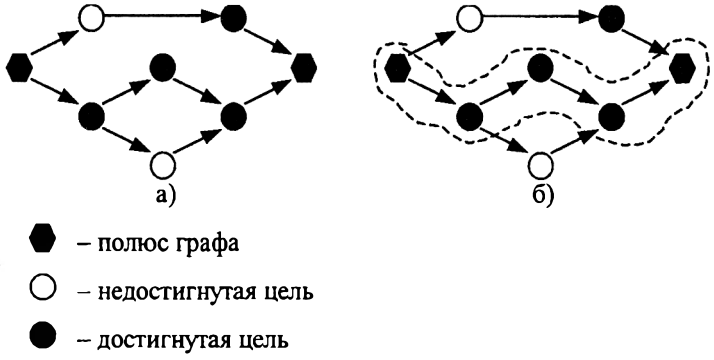


Рис.1. Пример графа планирования

Базу знаний решателя составляют следующие относительно независимые друг от друга части:

1. база стереотипов, являющихся основным проблемно-ориентированным средством поиска решения;
2. база задачи, которая содержит знания о геометрических объектах задачи и отношениях между ними;
3. база целей, хранящая знания об искомой цели и целях, добавленных в ГПР в ходе планирования;
4. дневник планирования, содержащий знания о состоянии процесса планирования.

В РГЗ используются два вида стереотипов: функциональные и управляющие. Функциональные стереотипы (ФС) представляют знания о способах решения геометрических задач. С их помощью производится уточнение целей и вывод новых фактов (более подробно

см. [Корухова и др., 94]). Управляющие стереотипы – это знания из области планирования. На их основе в решателе построено управление ходом решения и реализованы различные стратегии планирования.

Процесс планирования РГЗ можно разделить на следующие этапы:

1. Ввод условий задачи;
2. Циклическое применение методов планирования на различных шагах решения до тех пор, пока оно не будет получено, или возникнет тупик;
3. Выдача объяснения полученного решения.

В ходе планирования на каждом шаге осуществляется выбор одного из используемых в РГЗ методов планирования. При применении конкретного метода принимается решение о том, какой именно из способов действий применять (каким из реализованных способов выбирать текущую цель в ГПР, подлежащую уточнению, на основании какого критерия отбирать функциональные стереотипы для уточнения текущей цели или вывода новых фактов и т.д.). В этом выборе и заключается управление планированием, в зависимости от сделанного выбора решение задачи пойдет тем или иным путем.

Мы предлагаем производить этот выбор на основе знаний, представленных в виде *управляющих стереотипов* – при выполнении определенных условий на состояние процесса планирования активировать те или иные действия. *Управляющий стереотип* (УС) [Корухова и др., 2000] представляет собой именованный элемент базы знаний, который на основе ситуации, сложившейся в ходе решения задачи, предписывает выполнить описанные в нем действия. УС в базе знаний представлен в виде фрейма с пятью слотами:

Имя: уникальный идентификатор стереотипа;

Комментарий: содержательный текст, который может быть использован при объяснении процесса построения плана;

Ситуация: ссылка на описание ситуации в базе ситуаций;

Действие: последовательность операторов запроса к служебным компонентам РГЗ или операторов записи в дневник планирования;

Ценность: характеристика стереотипа, используемая при выборе одного УС из нескольких применимых.

УС обеспечивают простое добавление новых служебных компонент в решатель и гибкое управление их запуском. При добавлении новой компоненты в решатель нет необходимости изменять программу-администратор. Достаточно добавить в базу знаний описание ситуации, в условиях которой следует обратиться к компоненте, и УС, в действии которого есть запрос, обрабатываемый компонентой.

Связанная с управляющим стереотипом ситуация содержит требования на состояние дневника планирования перед запуском данного стереотипа. Дневник планирования – это специальная структура для хранения знаний о состоянии процесса планирования. Он представляется фреймом со следующими слотами:

1. Номер текущего шага;
2. Текущая цель;
3. Комментарий к текущему шагу;
4. Текущая стратегия планирования;
5. Уточнение текущей цели;
6. Текущий список примененных при прямом выводе формул и теорем;
7. Флаги:
8. тупика;
9. начала сеанса;
10. окончания сеанса;
11. завершения решения.

Слоты дневника допускают чтение, запись и различные проверки (на известность, на равенство, неравенство, $<$, $>$, $\leq$, $\geq$ – для слотов-чисел). Специфика дневника в том, что значения всех его слотов (кроме специальных – флагов) меняются с переходом на новый шаг планирования, при этом значения на предыдущих шагах сохраняются.

Примитивы работы с дневником:

- записать значение слота дневника;
- получить текущее значение слота;
- получить значение слота на предыдущем шаге.

На основании дневника работ производится распознавание ситуаций для управляющих стереотипов. Он играет роль схожую с базой задачи, содержимое которой анализируется при распознавании ситуаций для функциональных стереотипов.

Теперь перейдем к обсуждению алгоритма администратора решателя, который используется при управлении процессом планирования с использованием управляющих стереотипов. Он состоит из следующих шагов:

1. Начало сеанса работы;
2. Распознавание ситуаций для управляющих стереотипов на основании анализа слотов дневника планирования;
3. Если есть распознанные ситуации, то выполняется разрешение конфликта (выбор одного УС из нескольких применимых), иначе переход на шаг 6;
4. Выполнение действий выбранного УС;
5. Если не установлен флаг окончания сеанса, то переход на шаг 2;

6. Освобождение ресурсов, окончание сеанса.

На первом шаге производится инициализация сопоставителя ситуаций функциональных стереотипов, создается база задачи, база целей (в них сначала содержатся только родовые фреймы и ни одного фрейма-экземпляра) и дневник планирования (также пустой). Затем слоты дневника получают значения по умолчанию, и одновременно происходит процесс распознавания ситуаций в дневнике планирования.

На третьем шаге алгоритма администратора происходит разрешение конфликта – выбор одного из управляющих стереотипов, для которых распознаны ситуации. В качестве способа разрешения конфликта используется следующее правило: предпочтение отдается стереотипам, которые имеют бóльшую ценность, и ситуации которых имеют большее число условий.

После разрешения конфликта выполняются действия выбранного стереотипа. Признаком завершения работы решателя является флаг окончания сеанса, который может быть установлен при выполнении действий вызванного управляющего стереотипа. При установленном флаге осуществляется выход из цикла работы администратора, освобождение ресурсов, занимаемых решателем, и окончание работы. Выход из цикла происходит как в случае успешного завершения планирования, так и в случае обнаружения в решении тупика.

2. Реализация различных методов и стратегий планирования

В решателе реализовано несколько методов планирования. Принятие решения о том, какой метод использовать при построении решения задачи, мы называем выбором стратегии. Соответственно, стратегии делятся на два типа: чистые и смешанные. При использовании чистой стратегии все решение задачи от начала до конца производится одним и тем же методом. При смешанной стратегии методы чередуются или комбинируются каким-либо способом.

Идея ***метода обратной цепочки рассуждений*** состоит в том, чтобы последовательно проводить редукцию цели, поставленной в задаче, и подцелей, возникающих в ходе планирования, до тех пор, пока искомая цель не будет представлена совокупностью элементарных подцелей, достижение которых не составляет труда (см. Рис.2).

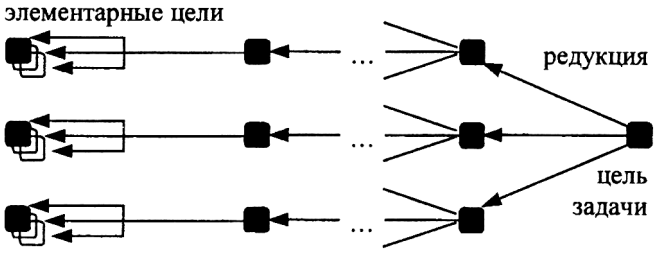


Рис.2. Редукция целей в методе обратной цепочки рассуждений

Действия метода обратной цепочки рассуждений на одном шаге планирования описываются следующим алгоритмом:

1. Среди неуточненных и недостигнутых целей в ГПР выбирается текущая цель (ТЦ) для уточнения.
2. Формируется активный контекст ТЦ (чтобы можно было опираться на условные факты – недоказанные узлы-цели, которые лежат слева от текущей цели на путях в графе, проходящих через нее).
3. Определяется набор функциональных стереотипов, пригодных для уточнения текущей цели.
4. Если подходящих стереотипов нет, то ТЦ заносится в «черный список», и осуществляется переход на следующий шаг планирования.
5. Найденные стереотипы применяются и уточняют цель подграфом, который стоится так, чтобы не возник «порочный круг» - цикл в ГПР.
6. Если задача решена, то устанавливается флаг окончания решения, иначе осуществляется переход на следующий шаг планирования.

Метод ***прямой цепочки рассуждений*** также как и метод обратной цепочки рассуждений относится к традиционным методам планирования. Отличие от описанного ранее метода состоит в том, что данный метод не использует разбиение целей на подцели. При прямой цепочке рассуждений осуществляется вывод новых фактов из условий задачи и фактов, полученных в ходе вывода, пока не будет получен факт, соответствующий поставленной в задаче цели (см. Рис.3).

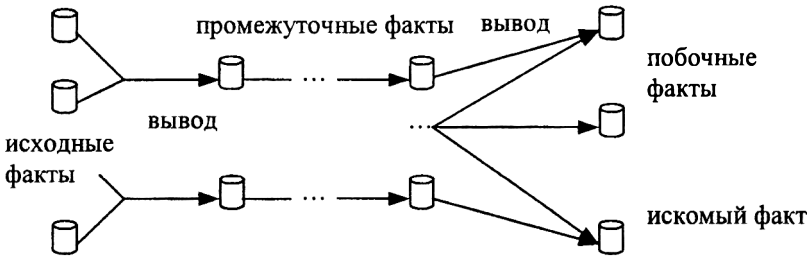


Рис 3. Вывод фактов при прямой цепочке рассуждений.

Алгоритм шага планирования этого метода таков:

1. Распознаются ситуации ФС, определяется набор ФС, пригодных для вывода новых фактов.
2. Если нет пригодных стереотипов, то метод терпит неудачу, устанавливается флаг тупика, и решение заканчивается.
3. Действия отобранных ФС применяются, и новые факты записываются в базу задачи.
4. Если добавлен факт, соответствующий искомой цели, то решение заканчивается, в противном случае осуществляется переход на следующий шаг решения.

Ассоциативное планирование – это новый метод планирования, предложенный при разработке РГЗ [Вакин и др., 1997]. Здесь мы ограничимся рассмотрением идеи метода.

При ассоциативном планировании, как и при планировании методом обратной цепочки рассуждений производится уточнение целей из ГПР. Методы отличаются способом построения уточнения. Уточнение текущей цели, получаемое при ассоциативном планировании, строится с использованием аналогии. Идея метода такова: если при решении задачи удается установить, что для применения некоторого стереотипа, уточняющего текущую цель, не хватает фактов в базе знаний, то следует осуществить вывод недостающих фактов и применить действия этого стереотипа.

Реализация ассоциативного планирования базируется на идее использования близких ситуаций (см. Рис.4).

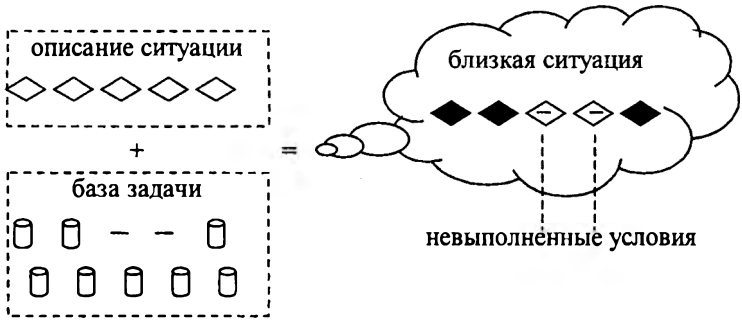


Рис. 4. Близкая ситуация

В ходе построения ассоциативного уточнения производятся следующие действия:

1. Производится поиск стереотипов, которые уточняют цели того же типа, что и текущая цель, но ситуации которых нераспознаны.
2. Из отобранных стереотипов для дальнейшего рассмотрения оставляются лишь те, ситуации которых можно сделать распознанными, добавив в базу задачи некоторые наборы фактов. Для каждого из таких стереотипов определяется конкретный набор недостающих фактов.
3. Оставшиеся стереотипы оцениваются согласно критериям «полезности», «доступности» и комбинированному критерию, для того чтобы избежать излишней сложности строящегося уточнения ТЦ.
4. Для каждого стереотипа, прошедшего все проверки, строится соответствующая ему часть уточнения. Создается цепочка целей, достижение которой приведет к добавлению в базу задачи недостающих фактов. Эта цепочка последовательно объединяется с цепочкой целей, добавляемой при выполнении действий стереотипа.
5. Построенные части уточнения объединяются параллельно и образуют искомое ассоциативное уточнение.
6. При вставке полученного уточнения в граф планирования из него удаляются части, приводящие к образованию циклов в графе.

Несколько слов о критериях отбора ситуаций. Условие «полезности» стереотипа заключается в том, что трудность его действия не должна превышать установленного предела. Выполнение данного условия гарантирует, что применение действия ФС не приведет к излишнему усложнению плана. Условие «доступности» ограничивает суммарную трудность целей, входящих в цепочку для вывода недостающих фактов. Вывод недостающих фактов должен быть априорно достаточно простым. При проверке данного условия учитывается тип недостающих фактов, поскольку от типа факта зависит трудность цели (найти площадь треугольника труднее, чем величину

его стороны). Комбинированный критерий позволяет совместно оценить «полезность» стереотипа и «доступность» связанной с ним ситуации. С помощью комбинированного критерия исключаются из рассмотрения стереотипы, для которых значения обеих характеристик близки к предельным.

Шаг ассоциативного планирования описывается алгоритмом, схожим с тем, что был приведен ранее для метода обратной цепочки рассуждений, надо лишь иметь в виду разницу в способе построения уточнения текущей цели.

Нужно отметить, что ассоциативное планирование не может использоваться как самостоятельный метод планирования, поскольку получаемое при этом методе уточнение обязательно содержит недостигнутые цели, подлежащие дальнейшему уточнению. Решение задачи может быть получено, только если ассоциативное планирование используется совместно с другими методами, что можно рассматривать как недостаток данного метода. В то же время ассоциативный метод имеет ряд преимуществ. В ситуации, когда при обратной цепочке рассуждений возникает тупик из-за отсутствия распознанных ситуаций, решение может быть продолжено ассоциативным методом, использующим стереотипы с нераспознанными ситуациями. Кроме того, при ассоциативном планировании решатель задач проводит более широкие рассуждения чем те, которые предусмотрены его базой функциональных стереотипов.

Для реализации смешанных стратегий в РГЗ есть два способа. Для стратегий, в рамках которых разные методы применяются на разных шагах планирования достаточно правильно описать ситуации для управляющих стереотипов выбора стратегии. Примерами таких стратегий являются: стратегия, на первом шаге которой применяется обратная цепочка рассуждений, на втором – прямая цепочка рассуждений и такое чередование продолжается, или стратегия, при которой в случае неудачи одного метода применяется другой (например, когда невозможно уточнить текущую цель – из базы задачи выводятся новые факты). В подобных случаях комбинирование различных методов не требует создания новых служебных компонент, достаточно создать набор управляющих стереотипов, что гораздо проще.

Другим видом комбинирования методов планирования является стратегии, на каждом шаге которых совместно применяются несколько методов (то есть, одновременно выполняется прямой и обратный ход, или объединяются уточнения текущей цели, полученные методом обратной цепочки рассуждений и ассоциативным планированием). Для таких стратегий создаются дополнительные служебные компоненты РГЗ, а также управляющие стереотипы, представляющие знания о таких стратегиях. В настоящее время

реализована одна стратегия такого типа – стратегия объединенного уточнения, при которой совместно применяется обратная цепочка рассуждений и ассоциативное планирование.

Рассмотрим пример решения задачи. Дано: ABCD – трапеция с основаниями AB и CD. Окружность O описана около ABCD. AC – диаметр. AB=20 см, AD=10 см. Найти: площадь трапеции ABCD.

При использовании прямой цепочки рассуждений решение осуществляется за четыре шага, при этом помимо фактов, необходимых для решения, выводятся факты, не имеющие значения для нахождения площади ABCD (в примере таковыми являются величины углов bac, acb, acd, cad). Решение, полученное прямой цепочкой рассуждений имеет следующий вид: *Угол adc – прямой, так как противолежащая ему в треугольнике ACD сторона AC является диаметром описанной вокруг ACD окружности O. Угол abc – прямой, так как противолежащая ему в треугольнике ABC сторона AC является диаметром описанной вокруг ABC окружности O. Угол bcd=180-abc=90, так как они внутренние односторонние при основаниях AB и CD трапеции ABCD. Угол bad=180-adc=90, так как они внутренние односторонние при основаниях CD и AB трапеции ABCD. Четырехугольник ABCD – прямоугольник, так как все его углы (bad, abc, bcd, adc) равны 90. Площадь прямоугольника ABCD=AB*AD = 20*10=200.*

При использовании смешанной стратегии состоящей в чередовании прямой и обратной цепочек рассуждений на разных шагах решения цель задачи достигается за пять шагов. На последнем шаге решения граф планирования имеет вид, изображенный на Рис. 5. Некоторые цели лежащие на решающем пути достигнуты при редукции, другие – при выводе новых фактов. Объяснение решения, построенное решателем, имеет вид: *Угол abc - прямой, так как противолежащая ему в треугольнике ABC сторона AC является диаметром описанной вокруг ABC окружности O. Трапеция ABCD с основаниями AB и CD равнобочная, так как около нее описана окружность O. BC=AD. Площадь треугольника ABC вычисляется как S=0.5*AB*BC*sin(abc)= 100. По теореме косинусов длина отрезка AC вычисляется через отрезки AB,BC и угол abc и AC=22.22.360679775. Угол adc – прямой, так как противолежащая ему в треугольнике ACD сторона AC является диаметром описанной вокруг ACD окружности O. По теореме Пифагора CD вычисляется как катет через гипотенузу AC и катет AD, и CD=20. Площадь треугольника ACD вычисляется как S=0.5*CD*AD*sin(adc)= 100. Площадь ABCD = площадь ACD + площадь ABC = =100 + 100 =200.*

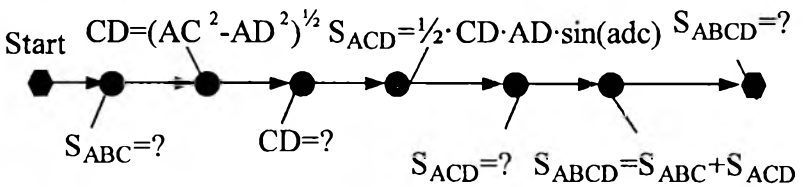


Рис. 5. Окончательный вид графа планирования при стратегии чередования прямой и обратной цепочек рассуждений

Перейдем к рассмотрению решения, полученного другой смешанной стратегией, при которой осуществляется объединение ассоциативного планирования и обратной цепочки рассуждений. На первом шаге будут получены два способа достижения поставленной в задаче цели, соответственно в графе планирования появятся два пути (см. Рис. 6). Нижний путь получен на основе близкой ситуации (при известных двух соседний сторонах для нахождения площади достаточно доказать, что ABCD является прямоугольником). При дальнейшем планировании происходит уточнение целей на нижнем пути, так как он является более дешевым. За четыре шага производится доказательство того, что все углы ABCD являются прямыми. По окончании шестого шага планирования на нижнем пути в графе не остается недостигнутых вершин, и решение заканчивается. Объяснение полученного решения совпадает с объяснением, полученным при прямой цепочке рассуждений. Следует отметить, что все факты, выведенные в ходе решения, имеют непосредственное отношение к достижению поставленной цели. По времени счета решение, рассмотренное последним, является самым коротким, так как при ассоциативном планировании и обратной цепочке рассуждений число фактов, анализируемых сопоставителем ситуаций ФС, значительно меньше, а на его работу приходятся основные временные затраты.

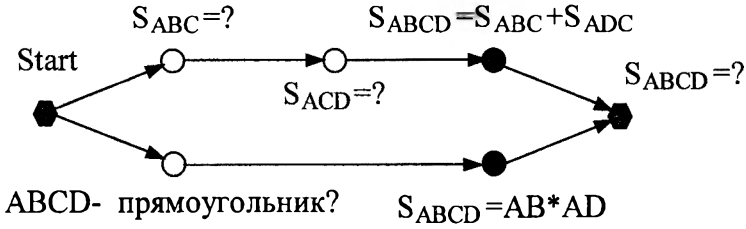


Рис. 6. Граф планирования после первого шага решения

Заключение

В работе рассмотрен новый метод планирования для решения сложных задач – ассоциативное планирование на основе близких ситуаций. Описана реализация экспериментального решателя, осуществляющего планирование задач дедуктивными и ассоциативными методами, использующего чистые и комбинированные стратегии.

Эксперименты с решателем позволили провести оценки применения различных стратегий в планировании решений задач. Так для приведенного примера применение смешанной стратегии планирования оказалось более эффективным (см. Таблицу 1).

Стратегия (вид стратегии)	Количество шагов решения	Количество выведенных фактов
Прямая цепочка рассуждений (чистая)	4	17
Чередование обратной и прямой цепочек рассуждений (смешанная)	5	10
Объединение ассоциативного планирования и обратной цепочки рассуждений (смешанная)	6	6

Таблица 1. Характеристики полученных решений

Литература

1. [Вакин и др., 1997] Вакин В. В., Корухова Л. С., Любимский Э. З., Малышко В. В. *Ассоциативные методы планирования решений сложных задач.* – М.: Препринт Института прикладной математики РАН, № 81, 1997 г.

2. [Корухова и Любимский, 1978] Корухова Л. С., Любимский Э. З. *Система планирования действий, основанная на некоторых естественных принципах – ПЛАР.* – М.: Препринт Института прикладной математики АН СССР. № 53. 1978.

3. [Корухова и др., 1994] Корухова Л. С., Любимский Э. З., Островский В. В. *Программирование на основе стереотипов.* – М.: Препринт Института Прикладной Математики РАН, № 18, 1994.

4. [Корухова и др., 2000] Корухова Л. С., Любимский Э. З., Малышко В. В. *Реализация стратегий планирования на основе управляющих стереотипов.* – М.: Препринт Института прикладной математики РАН, № 13, 2000 г.

5. [Ленат, 1984] Ленат Д. Б. *Искусственный интеллект.* Сб. "Современный компьютер", перевод на русский язык. – М.: Мир, стр.174. 1984.

Раздел IV
Интеллектуальный анализ данных

Кичигин Д.Ю.

О применении подходов Data Mining для обнаружения вторжений[1]

Введение

Под названием Data Mining понимается группа методов, направленных на извлечение скрытых закономерностей, «знаний» из больших объемов данных [2].

Одной из традиционных областей применения методов Data Mining, наряду с такими областями как маркетинг, анализ производственных процессов, телекоммуникации и многие другие [15], является обнаружение компьютерных вторжений. Целью систем обнаружения вторжений является выявление попыток или обнаружение уже совершенных вторжений. Существует несколько определений *вторжения(intrusion)*. В работе [1] вторжением в компьютерную систему, называется любое совершенное в отношении этой системы действие, если совершивший его объект, называемый злоумышленником, не имел права его совершать. В работе [4] вторжение определяется как любая деятельность, нарушающая целостность, доступность или конфиденциальность данных. Оба определения достаточно хорошо описывают то, с чем приходится бороться системам обнаружения вторжений.

Традиционно, системы обнаружения вторжений представляли собой экспертные системы [3]. Для выбора нужных признаков и описания правил использовались экспертные знания или интуитивные представления о работе сетей, операционных систем и методах вторжений. В настоящее время, в условиях сильно возросшей сложности защищаемых систем и изощренности злоумышленников, экспертное знание становится ограниченным и не настолько надежным чтобы полагаться только на него. Кроме того, возросший объем сканируемой информации потребовал эффективных методов ее

[1] Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект № 03-01-00745

обработки. Все это обусловило активное использование методов Data Mining в этой области.

В настоящей работе мы рассмотрим способы использования методов Data Mining в задаче выявления вторжений. В Главе «Системы обнаружения вторжений» мы рассмотрим особенности задачи обнаружения вторжений и основные подходы в организации систем обнаружения вторжений. В Главе «Применение методов Data Mining в задаче обнаружения вторжений» мы приведем обзор применения методов Data Mining к задаче обнаружения вторжений.

Системы обнаружения вторжений

Постановка задачи обнаружения вторжений включает следующие ограничения [1]: не рассматривается ситуация полного физического уничтожения системы. Предполагается что компьютерная система, как и система обнаружения вторжений продолжают функционировать в течении некоторого времени после атаки или ее попытки, так что они могут предупредить об этом. Также предполагается неприкосновенность самой системы обнаружения вторжений и ее данных. Эти проблемы являются классическими в компьютерной безопасности и достаточно хорошо изучены.

Первые системы обнаружения вторжений относятся к 70-м – началу 80-х гг. Это были off-line системы, работающие локально на отдельных компьютерах. Далее развитие систем происходило в сторону усовершенствования методов обнаружения вторжений, появилась on-line - реакция на проводящиеся вторжения. Областью «ответственности» систем стали распределенные компьютерные системы. Для современных систем обнаружения вторжений характерна работа в больших сетях и обработка больших объемов данных. Также современные системы обнаружения вторжений предпринимают on-line реакцию на вторжение с целью его предотвращения (intrusion prevention) либо уменьшения последствия вторжения.

Существуют два основных подхода к обнаружению вторжений. Первый, *Обнаружение Аномалий* (*Anomaly Detection*) основывается на построении модели, описывающей корректное состояние и/или допустимое поведение системы, и последующем выявлении опасных изменений состояния или опасного поведения системы. Это выявление основывается на анализе отклонений состояния или поведения системы от построенной модели. Другой подход называется *Обнаружением Злоупотреблений* (*Misuse Detection*) и заключается выявлении уже известных вторжений. Т.е. строятся шаблоны уже известных атак и

состояние и/или поведение системы сканируются на предмет совпадения с шаблонами.

Оба подхода появились еще в первых системах обнаружения вторжений и используются до настоящего момента. Каждый из них имеет свои достоинства и недостатки. Обнаружение злоупотреблений хорошо работает с известными вторжениями, но в то же время не чувствительно к новым, неизвестным атакам. Обнаружение аномалий, наоборот, более хорошо распознает неизвестные атаки, но проигрывает обнаружению злоупотреблений при работе с известными. Поэтому эти подходы являются взаимодополняющими и каждый из них используется и развивается до сих пор.

Обнаружение вторжений может быть *статическим* (*static*) или *динамическим* (*dynamic*). Статическое обнаружение вторжений анализирует статическое состояние компьютерной системы. Это может быть, например, размер и совокупность данных на диске. Динамическое обнаружение вторжений анализирует поведение системы. Поведение системы обычно описывается в виде последовательности *событий(events)*. Это могут быть, например, данные лог-файла, который создается операционной системой в процессе своего функционирования.

Обнаружение Аномалий

Статическое обнаружение аномалий подразумевает анализ тех частей программного обеспечения системы, которые должны оставаться неизменными. Другими словами статическое обнаружение аномалий проверяет неприкосновенность данных. В случае, если обнаруживается изменение таких данных, то генерируется предупреждение. Динамическое обнаружение аномалий выдает предупреждения, если были замечены отклонения от поведения системы. Во обоих случаях, во время построения модели системы подразумевается что поведение или состояние системы являются нормальными, т.е. не содержат вторжений.

Статическое обнаружение аномалий

Примером статического обнаружения аномалий является сигнатурный подход, при котором происходит кодирование нормального состояния системы в некоторое представление, называемое *сигнатурой(signature)*. Выбор способа построения сигнатуры выбирается с учетом того, чтобы разным состояниям системы с большой вероятностью соответствовали разные значения сигнатуры. После этого построенная сигнатура периодически сравнивается с сигнатурой текущего состояния системы, построенной по тем же правилам. Если они не совпадают, значит наблюдается аномалия, которая может означать вторжение или сбой в системе. Кроме непосредственно сигнатур часто используются некоторая дополнительная информация о структуре проверяемых объектов. Например для лог файла такой информацией является его размер: уменьшение размера лог файла с большой вероятностью может означать вторжение, тогда как увеличение скорее всего является нормальной ситуацией.

Другим интересным решением является подход получивший название Self/Nonself [14]. Состояние компьютерной системы рассматривается как одна строка данных. Из этой строки выделяются подстроки фиксированной длины K, которые образуют множество, называемое Self, соответствующее нормальному состоянию системы. После этого строится множество подстрок *Nonself* также фиксированной длины K, которые *не* соответствуют нормальному состоянию системы. Это множество строится следующим образом: генерируется множество U, содержащее всевозможные подстроки длины K, и Nonself полагается равным U \ Self. Далее, мониторинг системы состоит из периодического сравнения подстрок длины K, соответствующих текущему состоянию системы со строками из множества Nonself: если обнаруживаются совпадения то фиксируется аномалия.

Динамическое обнаружение аномалий

Для динамического обнаружения вторжений характерно то что поведение системы может достаточно широко варьироваться. Поэтому для динамического обнаружения аномалий часто используются статистические методы. Строится вероятностная модель поведения системы и отклонение определяется в статистических терминах. Величина отклонения позволяет судить о вероятности того что наблюдается аномалия. Для отделения нормального поведения системы от аномального используются *пороги* (*thresholds*), которые определяются вручную.

Другим подходом является построение модели текущего состояния системы и периодическое сравнение ее с моделью нормального поведения. Вычисляется «разница» между моделями. Если разница слишком велика (больше некоторого заданного порога) то фиксируется аномальное поведение системы.

Еще одним интересным подходом является подход, разработанный в Университете г. Нью-Мехико (University of New Mexico) [7,8,9]. Подход основывается на мониторинге последовательностей вызовов системных функций операционной системы UNIX, осуществляемых привилегированными процессами во время своего выполнения. Такие программы обладают более обширными правами чем обычные пользователи(даже запустившие их), поэтому могут использоваться для осуществления вторжений. Например, если такая программа имеет некоторую уязвимость, то запустивший ее пользователь, используя эту уязвимость, может получить доступ к системе с более обширными правами чем имел до этого. Этот подход является результатом применения техники Self/Nonself, описанной в статическом обнаружении аномалий, к трассам программ. Работа каждой программы рассматривается как последовательность совершаемых ею вызовов системных функций операционной системы. Эта последовательность разбивается на подстроки фиксированной длины и далее строится множество Self соответствующее нормальному поведению процесса. Во время выявления аномалий строки, взятые из трассы текущего поведения системы ищутся в множестве Self. Если в множестве Self такой строки нет, то фиксируется «несовпадение». Количество несовпадений определяет степень аномальности текущего выполнения процесса.

Наибольшую сложность в динамическом выявлении аномалий представляет собой аккуратное построение модели нормального поведения системы. Модель может быть построена искусственно, моделируя поведение системы во всех возможных ситуациях, либо используя информацию о поведении системы в обычном режиме на протяжении довольно длительного периода времени.

Каждый из способов имеет свои достоинства и недостатки. «Искусственный» сбор данных дает более полную картину о возможном поведении системы. Данные, собранные во время обычного функционирования системы более хорошо показывают как ведет себя система в реальных условиях.

Обнаружение злоупотреблений

Обнаружение злоупотреблений имеет своей целью выявление попыток уже известных способов вторжений. В данном случае информация о предыдущем нормальном поведении или состоянии системы не рассматривается. Здесь также существуют статический и динамический подходы. Классическим примером системы выявления злоупотреблений являются антивирусы. Антивирусы поддерживают базу данных, содержащую шаблоны кодов вируса и потом ищут эти шаблоны в системе. В случае удачи антивирусы сообщают о наличии вируса.

Динамическое обнаружение злоупотреблений основано на периодическом сравнивании поведения системы с уже известными сценариями вторжений. *Сценарием вторжения* (*intrusion scenario*) называется описание известного способа вторжения. Обычно сценарий вторжения определяется в терминах последовательности действий, результатом которых является вторжение. В данном случае эффективность обнаружения вторжений зависит от качества описания сценариев.

Системы динамического обнаружения злоупотреблений различаются в основном способом описания вторжений. Существует два основных подхода. В первом случае для этих целей используются правила. Системы обнаружения вторжений представляют собой экспертные системы, где сценарии вторжения закодированы в набор правил. Эти правила отражают частично упорядоченную последовательность действий, приводящих к вторжению. Вторым подходом для описания сценариев вторжений является использование описаний состояний системы. Сценарии вторжений описываются с помощью направленных графов, где вершинам соответствуют состояния системы, а ребрам – действия. Состояние системы могут

включать в себя описание работающих процессов, количество работающих на системе пользователей и т.п. Действия провоцируют переход системы из одного состояния в другое. Граф, описывающий сценарий вторжения состоит из начального состояния, т.е. состояния системы перед началом вторжения, нескольких промежуточных состояний и завершающего состояния которое сигнализирует о вторжении.

Применение методов Data Mining в задаче обнаружения вторжений

Хотя Data Mining объединяет под своим названием множество методов из различных дисциплин (базы данных, статистический анализ, визуализация данных, высокопроизводительные вычисления, искусственный интеллект, нейронные сети, обработка образов и сигналов и т.д.) и позволяет решать достаточно широкий круг задач, из этих задач можно выделить 6 основных, к которым сводятся все остальные [2]:

1. Описание данных – задача нахождения некоторого компактного описания имеющихся данных.
2. Анализ ассоциаций – поиск модели, описывающей зависимости между значениями атрибутов данных.
3. Классификация и Прогнозирование –
   - Классификация – распределение данных в некоторый предопределенный набор классов.
   - Прогнозирование – предсказание неизвестных значений атрибутов данных по известным .
4. Кластерный анализ – обработка данных с целью выделения заранее неизвестных групп «похожих» данных (кластеров).
5. Выявление исключений – поиск данных, которые ведут себя «не так как все», являются «исключениями из правил».
6. Эволюционный анализ – выявление закономерностей меняющегося со временем поведения объектов.

В области обнаружения вторжений методы Data Mining как правило применяются для автоматической подборки признаков и построения моделей, использующихся для выявления вторжений. Модели нормального поведения и модели вторжений могут быть построены автоматически, основываясь на данных о предыдущем поведении системы, и потом использованы для обнаружения будущих вторжений. При этом задача обнаружения вторжений может быть представлена как задача классификации, т.е. требуется сделать прогноз

относительно новых данных – принадлежат ли они к классу вторжений, классу нормального поведения или состояния системы или они принадлежат к классу аномального поведения системы (возможно являются неизвестным типом вторжения).

В последние годы было проведено много исследований в области применения методов Data Mining к задаче обнаружения вторжений.

Анализ связей

Один из известных подходов в этой области основан на анализе связей [4]. Суть подхода состояла в построении системы правил, описывающих поведение системы. Авторами была показана эффективность такого подхода как для выявления злоупотреблений так и для выявления аномалий. Этот подход был применен для анализа последовательностей системных вызовов, совершаемых программой и для анализа сетевого трафика.

В случае выявления злоупотреблений выбор исходных данных основывался на результатах работ, проведенных в университете New Mexico, где показывалось что последовательности системных вызовов, совершенных процессом во время своего выполнения, являются хорошим материалом для обнаружения вторжений [7,8,9]. Далее к таким последовательностям применялся анализ ассоциаций, являющийся одним из подвидов анализа связей, результатом которого была система правил, описывающих наиболее часто встречающиеся сочетания системных вызовов в последовательностях. Для обнаружения злоупотреблений на вход алгоритму поставлялись последовательности системных вызовов, соответствующих атакам и далее построенная система правил применялась для анализа последующего поведения системы. Для выявления аномалий система правил строилась на основе данных о нормальном поведении программы и далее использовалась для анализа «нормальности» последующего поведения программы.

При выявлении аномалий в качестве исходных данных рассматривался сетевой трафик, проходящий между различными узлами сети. При помощи алгоритма *частых эпизодов (frequent episodes)* строилась система правил, описывающая наиболее часто встречающиеся эпизоды(последовательности) соединений, которые впоследствии использовались для характеристики атак или нормальной сетевой активности.

Поиск исключений

Методы поиска исключений также могут успешно применяться в обнаружении вторжений. В подходе, описанном в работе [5], используется метод поиска исключений, представляющий собой комбинацию метода нечеткой кластеризации и потенциальных функций для обнаружения аномалий. Исходные данные, представляющие собой поток сетевых соединений, с помощью потенциальной функции отображаются в пространство характеристик высокой размерности, где решается задача нечеткой кластеризации. В результате каждому соединению ставится в соответствие специальная характеристика - нечеткая степень «типичности», показывающая насколько данное соединение можно считать аномальным.

Нейронные сети

Другим подходом является использование аппарата нейронных сетей. Нейронная сеть представляет собой совокупность узлов и дуг. Каждый узел имеет несколько входов и один выход и реализует некоторую функцию. Дуги связывают узлы друг с другом: передают информацию с выхода одних узлов на входы других. Существуют три типа узлов: входные узлы, получающие на свои входы информацию от внешних источников, выходные узлы, передающие наружу результаты работы сети и промежуточные узлы, сюда попадают все остальные узлы. Таким образов вся нейронная сеть представляет собой единый механизм, решающий некоторую определенную задачу: получает данные на вход и по определенным правилам формирует результат.

Нейронные сети могут быть разделены на два класса по типу обучающего алгоритма [6]. Первый класс соответствует сетям, обучающимся с помощью алгоритмов контролируемого обучения. Во второй класс входят сети, обучающиеся с помощью алгоритмов неконтролируемого обучения (в русскоязычной литературе также встречаются названия алгоритм с учителем и алгоритм без учителя). Типичным представителем сетей первого класса является многослойный персептрон(Multi-Level Perceptron) [16]. Типичным представителем сетей второго класса являются сети Кохонена (Self Organizing Maps). Сети Кохонена строят отображение элементов исходных данных в заранее неизвестные кластеры, которые потом используются при классификации вновь поступающих данных.

В обнаружении вторжений используются оба класса сетей. Причем они применяются как в обнаружении аномалий [13] так и в обнаружении злоупотреблений [12,13].

Скрытые Марковские Модели

В работе [11] авторами было показано применение аппарата Скрытых Марковских Моделей(СММ) для обнаружения вторжений. Суть подхода состояла в построении Скрытой Марковской Модели, описывающей нормальное поведение системы. Модель строилась на основе наблюдаемой последовательности событий в системе. Далее рассматривалась последовательность событий последующего функционирования системы и вычислялась вероятность того что наблюдаемая последовательность могла быть порождена построенной Марковской моделью. Вычисленная вероятность определяла степень аномальности последующего поведения системы.

Этот подход был применен для анализа последовательностей команд, запускаемых пользователем в операционной системе Unix [10]. Событиями являлись имена программ и их параметры. Степень аномальности последовательности событий показывала степень аномальности поведения пользователя. Поведение пользователя с высокой степенью аномальности объявлялось подозрительным, т.е. потенциально могло содержать враждебные действия.

Заключение

В настоящей статье было рассмотрено применение методов Data Mining для выявления компьютерных вторжений. В статье дан краткий обзор современных подходов, используемых при построении систем обнаружения вторжений. Сформулированы достоинства и недостатки этих подходов. Рассмотрен вопрос применения методов Data Mining в задаче применения вторжений и рассмотрены различные варианты успешного применения методов Data Mining в системах обнаружения вторжений.

Литература

1. A.K. Jones and R.S. Sielken, Computer System Intrusion Detection: A Survey, tech report, Computer Science Dept., University of Virginia, 2000.

2. Jiawei Han и Micheline Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, 2000

3. W.Lee, "Applying Data Mining to Intrusion Detection: the Quest for Automation, Efficiency, and Credibility", **ACM SIGKDD Explorations Newsletter,** Volume 4 , Issue 2 (December 2002) Pages: 35 – 42, 2002

4. W. Lee and S. Stolfo, "Data Mining Approaches for Intrusion Detection," Proc. 1998 7th USENIX Security Symposium, 1998. San Antonio, TX.

5. М.И.Петровский. Алгоритмы выявления исключений в системах интеллектуального анализа данных. // Журнал РАН «Программирование», 2003, №4, сс. 66-80.

6. Jean-Philippe Planquart, Application of Neural Networks to Intrusion detection, SANS Institute InfoSec Reading Room, Intrusion Detection July 29, 2001 (http://www.sans.org/rr/papers/30/336.pdf)

7. S. Forrest, S.A. Hofmeyr, A. Somayaji and T.A. Longstaff, "A Sense of Self for Unix Processes", IEEE Symposium on Computer Security and Privacy, Los Alamos, CA, pp.120-128, 1996.

8. S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. Journal of Computer Security, 6:151–180, 1998.

9. C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting Intrusions Using System Calls: Alternative Data Models," IEEE Symposium on Security and Privacy, Oakland, CA, 1999, IEEE Computer Society, pp. 133-145.

10. Lane, T. 1999: Hidden Markov Models for Human/Computer Interface Modeling. In: Proceedings of the IJCAI-99 Workshop on Learning about Users (1999) 35-44.

11. Lane, T. and C. E. Brodley: 1997, 'Sequence Matching and Learning in Anomaly Detection for Computer Security'. In: AAAI Workshop: AI Approaches to Fraud Detection and Risk Management. pp. 43–49.

12. Cannady, J. (1998). Neural Networks for Misuse Detection: Initial Results. Proceedings of the Recent Advances in Intrusion Detection '98 Conference, 31-47.

13. Ghosh and A. Schwartzbard, "A study in using neural networks for anomaly and misuse detection", 8th USENIX Security Symposium, pp. 141-151, 1999.

14. S. Forrest, A.S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In IEEE Symposium on Research in Security and Privacy, pages 202-212, Oakland, CA, May 16-18 1994. Springer-Verlag.

15. Fayyad, U., G. Piatetsky-Shapiro, and P. Smyth, From Data Mining to Knowledge Discovery in Databases. AI Magazine, 1996. 17: p. 37--54.

16. Заенцев И.В. Нейронные сети: основные модели. Уч. пособие.-Воронеж, 1999.-76 с.

Герасимов С.В.
Алгоритмы поиска ассоциативных правил[1]

Поиск ассоциативных правил [1] является одной из задач интеллектуального анализа данных (data mining) наряду с такими задачами как прогнозирование, классификация, кластеризация и выявление исключений.

Ассоциативные правила позволяют находить скрытые закономерности между атрибутами анализируемых объектов. По типу и особенностям атрибутов анализируемых объектов ассоциативные правила подразделяются на:

1. Булевы
2. Числовые
3. Обобщенные

Булевы ассоциативные правила [2] оперируют с логическими значениями атрибутов объектов. *Числовые ассоциативные правила* [5] - с объектами, атрибуты которых обладают числовыми значениями (как упорядоченными, так и значениями-перечислениями). *Обобщенные ассоциативные правила* [4] используются в тех случаях, когда атрибуты объектов составляют иерархическую структуру.

Классическим примером применения теории булевых ассоциативных правил является *задача анализа рыночной корзины* [6] (market basket analysis). В терминах этой задачи будут приведены определения, которые могут быть перенесены на теорию ассоциативных правил в целом.

Анализ рыночной корзины

Пусть имеется база данных, содержащая информацию о покупках, сделанных посетителями супермаркета. Каждая транзакция базы данных соответствует одной покупке и содержит информацию о том, какие товары из ассортимента супермаркета вошли в данную покупку.

Определение $_1, i_2, \ldots, i_n\}$ – множество наименований товаров, составляющих ассортимент супермаркета. D – множество транзакций. Каждая транзакция T – булев вектор (t[1], t[2], ..., t[n]) значений атрибутов из I. Значение "true" в j-й позиции вектора будет означать,

---

[1] Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект № 03-01-00745

что товар i_j присутствует в данной транзакции, значение "false" – что товар i_j не вошел в данную покупку.

Таким образом анализируемыми объектами задачи являются – покупки. Атрибутами анализируемых объектов является n индикаторов вхождения товаров из ассортимента в покупку.

Следует обратить внимание на то, что задача анализа рыночной корзины является задачей поиска именно булевых ассоциативных правил, поскольку в данном случае транзакция базы данных не содержит информации о количественных характеристиках покупки, например, о числе единиц приобретенного товара.

Определение *Транзакция* T *поддерживает множество атрибутов* (наименований товаров) $X \subseteq I$, если для каждого i_j, входящего в X соответствующее значение t[j] из T равно "true".

Определение *Булевым ассоциативным правилом* (далее просто *ассоциативным правилом*) называется упорядоченная пара множеств атрибутов (X, Y), где $X \subset I$ и $Y \subset I$ и $X \cap Y = \emptyset$. В качестве обозначения ассоциативного правила обычно используется символ импликации: X=>Y

Следующие два определения составляют т.н. показатель «интересности» ассоциативного правила

Определение *Поддержка (support)* ассоциативного правила X=>Y[1] – доля транзакций базы данных, поддерживающих множество атрибутов $X \cup Y$[2]. support(X=>Y)=P($X \cup Y$).

Определение *Достоверность (confidence)* ассоциативного правила X=>Y – доля транзакций, поддерживающих Y, среди транзакций, поддерживающих X. confidence(X=>Y)=P(Y|X)

Определение *Задача поиска ассоциативных правил.* Даны пороговые значения «интересности» правил: $support_{min}$, $confidence_{min}$. Требуется найти все ассоциативные правила, для которых одновременно выполняются условия: support(X=>Y)>=$support_{min}$ и confidence(X=>Y)>=$confidence_{min}$. Найденные правила называются *интересными правилами.*

Таким образом задача поиска ассоциативных правил заключается в поиске правил, удовлетворяющим фиксированным порогам поддержки и достоверности. Причем оба эти показателя:

---

[1] Корректней было бы говорить не о поддержке ассоциативного правила, а о поддержки множества атрибутов, входящих в него

[2] Таким образом определенную поддержку иногда называют процентной поддержкой, т.к. речь в данном определении идет о доли или процентном отношении транзакций, поддерживающих множество атрибутов, относительно всего числа транзакций. Само число транзакций, поддерживающих множество атрибутов называется абсолютной поддержкой.

поддержка и достоверность являются значимыми при отборе интересных правил. Поддержка позволяют судить о статистической обоснованности правила, достоверность – о статистике его выполнимости.

Важнейшим вопросом является вопрос выбора пороговых констант $support_{min}$, $confidence_{min}$ в каждой конкретной практической задаче. При занижении значения $support_{min}$ будут получены правила, которые не являются интересными, т.к. давно известны экспертам данной предметной области. При слишком низком значении $confidence_{min}$ многие из полученных правил не будут достоверными. При больших значениях порогов велика вероятность потери многих интересных правил. Результаты практического применения в различным областях свидетельствует о том, что действительно интересные и заранее неизвестные правила обнаруживаются при относительно небольшом значении порога поддержки.

Согласно формулировке задачи правила, полученные в результате, должны удовлетворять двум условиям: $support(X=>Y)>=support_{min}$ и $confidence(X=>Y)>=confidence_{min}$. Соответственно большинство алгоритмов поиска ассоциативных правил состоит из двух фаз:

1. 1-я фаза. Поиск наборов атрибутов, являющихся часто встречаемыми (удовлетворяющих условию $support(X=>Y)>=support_{min}$).
2. 2-я фаза. Генерация интересных правил из часто встречаемых наборов атрибутов (удовлетворяющих условию $confidence(X=>Y)>=confidence_{min}$)

Далее подробно рассматриваются наиболее популярные алгоритмы поиска ассоциативных правил Apriori и FP-деревья.

Apriori

Алгоритм был разработан сотрудниками исследовательского центра IBM Р. Агравалем (R. Agrawal) и Р. Срикантом (R. Srikant).

Предполагается, что множество атрибутов объектов I упорядочено, т.е. $i_1 < i_2 < i_3 < \ldots < i_{n-1} < i_n$.

Алгоритм использует следующую Лемму 1 о немонотонности[1][2]:

---

[1] Алгоритм использует частный случай леммы о немонотонности: множество атрибутов мощности k не будет часто встречаемым, если не является часто встречаемым хотя бы одно его подмножество мощности (k-1).

Лемма 1 (о немонотонности) множество атрибутов часто встречаемо только в том случае, когда часто встречаемыми являются все его подмножества.

Поиск часто встречаемых наборов атрибутов

L_1 = {часто встречаемые наборы из одного атрибута }
for(k = 2; L_{k-1} != Ø; k++) {
 C_k = apriori-gen(L_{k-1}); // Генерация k-элементных наборов-кандидатов
 forall T ∈ D {
  C_T = subset(C_k, T); /* Наборы-кандидаты, поддерживаемые
  транзакцией T */
  forall c ∈ C_T {
   c.count++;
  }
 }
 L_k = {c ∈ C_k | c.count >= $support_{min}$ }
}
Answer = $\bigcup_k L_k$;

На 1-й итерации генерируются одноэлементные часто встречаемые наборы атрибутов. Каждая последующая k-я итерация разбивается на 2 этапа:

1. Этап 1. Генерация множества k-элементных наборов-кандидатов по множеству (k-1)-элементных часто встречаемых наборов атрибутов с помощью функции apriori-gen
2. Этап 2. Подсчет поддержки для полученных кандидатов; отсеивание наборов, не удовлетворяющих порогу поддержки

Этап 1. Функция apriori-gen

Функция apriori-gen выполняется в 2 шага.

1-й шаг «Объединение» («Join»)

Его семантику можно сформулировать с помощью следующего SQL-запроса:
INSERT INTO C_k
SELECT p.i_1, p.i_2, …, p.i_{k-1}, q.i_{k-1}
FROM L_{k-1} p, L_{k-1} q

WHERE $p.i_1 = q.i_1$ AND $p.i_2 = q.i_2$ AND … AND $p.i_{k-2} = q.i_{k-2}$ AND $p.i_{k-1} < q.i_{k-1}$;

Запрос позволяет из (k-1)-элементных наборов получить k-элементные наборы[1]. После этого выполняется шаг «Очистка»

2-й шаг. «Очистка» («Prune»)

```
forall c ∈ Ck {
 forall (k-1)-элементных подмножеств s множества c {
  if(s ∉ Lk-1) {
   delete c from Ck;
  }
 }
}
```

Для каждого сгенерированного на 1-м шаге набора-кандидата происходит проверка принадлежности всех его (k-1)-элементных поднаборов множеству часто встречаемых (k-1)-элементных наборов (в соответствие с Леммой 1 о немонотонности)[2]. Наборы-кандидаты, не прошедшие проверку отбрасываются.

Этап 2. Подсчет поддержки

На этапе 2 происходит сканирование базы. Для каждой транзакции определяется какие k-элементные наборы-кандидаты она поддерживает. Для каждого из таких наборов счетчик поддержки увеличивается на 1. После выполнения сканирования из множества наборов-кандидатов отбрасываются наборы, не удовлетворившие порогу поддержки.

Авторы алгоритма предложили выполнять подсчет поддержки располагая наборы-кандидаты C_k в ХЕШ-дереве. Дерево организовано по следующим правилам. Внутренней вершиной дерева является ХЕШ-таблица, листья дерева – содержат наборы атрибутов и счетчики поддержки. Лист дерева способен содержать ограниченное число наборов-кандидатов.

---

[1] Причем семантика запроса позволяет избежать дублирования наборов и использует условие упорядоченности атрибутов

[2] Точнее всех поднаборов кроме двух: набора, включающего в себя все элементы C_k кроме последнего и набора, включающего в себя все элементы C_k кроме предпоследнего. Частая встречаемость этих двух поднаборов гарантируется семантикой запроса на шаге «Объединение»

Дерево строится каждый раз заново перед подсчетом поддержки для кандидатов. На 1-м слое дерева хешируются 1-е элементы наборов атрибутов, на 2-м – 2-е и т.д. Кроме самих наборов-кандидатов узлы-листья содержат текущее значение поддержки.

Во время сканирования базы очередная транзакция «пропускается» через дерево начиная от корня. В корневой вершине вопрос о том по какой ветке двигаться дальше решается с помощью применения ХЕШ-функции к первому по порядку атрибуту, входящему в транзакцию[1], в ХЕШ-таблицах, находящихся в узлах k-го уровня, - к k-му. В итоге транзакция эффективно «доходит» до нужного узла, у которого происходит инкрементирование значения поддержки.

Генерация интересных правил

После построения часто встречаемых наборов, выполненного на 1-й фазе алгоритма, происходит генерация правил, удовлетворяющих порогу достоверности. Эта задача гораздо менее трудоемкая. Для ее решения необходимо выполнить перебор всевозможных правил, которые можно построить из каждого часто встречаемого набора. Причем перебор можно сократить, используя следующую лемму 2[2].

Лемма 2 Пусть $X \cup Y$ – часто встречаемый набор атрибутов, $X = \{x_1, x_2, \ldots, x_t\}$, $Y = \{y_1, y_2, \ldots, y_s\}$. Тогда если правило $X \Rightarrow Y$ не удовлетворяет порогу достоверности, т.е. $confidence(X \Rightarrow Y) < confidence_{min}$, то и правило $\{x_1, x_2, \ldots, x_{t-1}\} \Rightarrow \{x_t, y_1, y_2, \ldots, y_s\}$ не будет удовлетворять условию достоверности.

Используя утверждение Леммы 2 простейший *алгоритм генерации правил* может выглядеть так[2]:

```
Rules=Ø; // Множество-результат
forall  Z∈ Answer AND |Z| >=2 {
 // Z={z1, z2, …, zk}
 generate-rule(Z, Ø);
}
gen-rule(Z, Y) {
 // Z – набор, содержащий все атрибуты транзакции
 /* Y={y1, y2, …, ys} - набор атрибутов правой части текущего
проверяемого правила *.
 if(Y!=Ø) {
  if(confidence(Z\Y=>Y) >= confidencemin) {
```

---

[1] К первому по порядку элементу транзакции, чье значение t[i] = «true»

[2] В оригинальном варианте авторами Apriori предлагается несколько иной усовершенствованный алгоритм генерации правил

```
    Rules ∪ = {Z\Y=>Y};
   } else {
    return;
   }
  }
  if(s >= |Z| - 1) {
   return;
  }
  // Пусть y_s = z_j
  for(i = j; i <= k; i++ ) {
   gen-rule(Z, {y_1, y_2, …, y_s, z_i});
  }
}
```

FP-дерево

В 2000 появился принципиально новый, по сравнению с Apriori, алгоритм. Он был предложен J. Han, J. Pei и Y. Yin в [3]. Концепция алгоритма – обнаружение часто встречаемых наборов без процедуры генерации наборов-кандидатов.

Основная вычислительная нагрузка в алгоритмах семейства Apriori ложится на два действия, выполняемые на каждой итерации:

Формирование множества наборов-кандидатов

Сканирование базы данных при подсчете поддержки

К примеру, при работе шага "Join" функции apriori-gen для k=100 может быть сгенерировано более $2^{100} \approx 10^{30}$ (!) наборов-кандидатов.

Новый алгоритм избавляет от необходимости создания множества кандидатов предлагая компактную структуру для эффективного хранения информации о всех часто встречаемых наборах, называемую *деревом часто встречаемых наборов* (frequent pattern tree или FP-tree). Алгоритм выполняется в 2 шага:

Построение FP-дерева

Генерация часто встречаемых наборов по FP-дереву

Шаг 1. Построение FP-дерева

Ниже приводится описание FP-дерева и алгоритм его построения.

Опр *FP-дерево* состоит из вершины, помеченной «null» и множества дочерних поддеревьев, а также набора часто встречаемых атрибутов. Каждый узел дочернего поддерева содержит три поля:

идентификатор атрибута item-name $\in$ I

количество транзакций count, поддерживающих ветвь FP-дерева, начинающуюся от корня и заканчивающуюся данным узлом

указатель node-link на следующий элемент FP-дерева, обладающий тем же item-name; если больше в дереве нет узлов с таким же item-name значение поля равняется null[1]

Алгоритм построения FP-дерева

База данных D сканируется 1 раз. Формируется набор F часто встречаемых атрибутов. В наборе войдут сами атрибуты и значения поддержки. Элементы в наборе упорядочиваются по невозрастанию значения поддержки.

Создается дерево с единственной вершиной-корнем root, обладающей меткой «null». Далее осуществляется сканирование всех транзакций базы данных. Для каждой транзакции T выполняются следующие действия:

Из набора F выбираются элементы (с сохранением порядка), которые поддерживаются транзакцией T. Обозначим этот набор как объединение [p|P], где p – 1-й по порядку атрибут в наборе, P – оставшиеся атрибуты набора. Происходит вызов процедуры добавления набора атрибутов в дерево insert_tree([p|P], root).

Семантика процедуры insert_tree([q|Q], K), где 1-й параметр – набор атрибутов, K-корень одного из поддеревьев в FP-дереве, такова:

Если у K существует дочерняя вершина N, такая, что N.item-name = q.item-name, то значение N.count увеличивается на 1, иначе в дереве создается новая вершина N с N.count=1 и присоединяется в качестве дочерней вершины к корню дерева root. В любом случае вершина N попадает в список вершин, относящихся к атрибуту q. Если набор Q не пуст далее происходит рекурсивный вызов insert_tree(Q, N).

Процесс построения FP-дерева требует всего двух сканирований базы данных: на 1-м проходе определяются часто

---

[1] Другими словами кроме привычных дуг, связывающих предков и потомков в дереве, узлы FP-дерева являются элементами специальных списков. Для каждого часто встречаемого атрибута создается список, в который при построении дерева последовательно попадут все узлы дерева, обладающие item-name, совпадающим с данным атрибутом.

встречаемые атрибуты, на 2-м происходит непосредственно создание дерева. Сложность добавления транзакции T в FP-дерево есть O(<число часто встречаемых атрибутов, поддерживаемых транзакцией T>)

Шаг 2. Генерация часто встречаемых наборов по FP-дереву

Алгоритм

```
Answer= Ø; // Множество-результат
call FP-growth(FP-tree, null);
FP-growth (Tree, α )
{
 if (Tree содержит единственный путь P) {
  для каждой комбинации вершин β в пути P do {
   поместить набор β ∪ α с поддержкой min по всем вершинам, входящим в β
(support) в Answer;
  }
 } else {
  Для каждого a_i ∈ набору часто встречаемых атрибутов[1] do {
   поместить набор β = a_i ∪ α с поддержкой = a_i.support в Answer;
   построить базовое множество ветвей для a_i;
   по нему построить условное FP-дерево Tree' ;
   if(Tree' != Ø) {
    FP-growth(Tree', β);
   }
  }
 }
}
```

Определение *Базовым множеством ветвей* в дереве Tree для атрибута a_i называется множество путей в дереве Tree, начинающихся от корня и ведущих узлам с item-name равным a_i[2]. Значение count у вершин, входящих в путь, инициализируется значением count у соответствующего узла с item-name равным a_i.

Определение *Построение условного FP-дерева.* Сначала происходит подсчет значения поддержки для всех атрибутов, входящих в в базовое множество ветвей. Затем для атрибутов, удовлетворивших порогу поддержки строится FP-дерево.

---

[1] Набор включает в себя те часто встречаемые атрибуты, которые присутствуют в дереве Tree

[2] Сам узел с item-name равным a_i, не включается в путь

Генерация правил может осуществляться с помощью любого алгоритма, например, с помощью описанного ранее для Apriori.

Заключение и выводы

Рассмотренные два алгоритма являются на сегодняшний день основными алгоритмами выявления ассоциативных правил в том смысле, что все остальные алгоритмы выявления ассоциативных зависимостей, используемые на практике, являются развитием их идей. В частности алгоритм AprioriTid [2] базируется на Apriori, но не требует сканирования всей БД на каждой итерации.

Как показано в [7], трудно дать аналитическую сравнительную оценку сложности обоих алгоритмов, не делая никаких предположения об особенностях данных, хранящихся в БД. Однако результаты тестов на различных видах искусственно сгенерированных наборов показывают, что FP-дерево существенно обгонят Apriori по скорости работы при значениях порога поддержки, близких к нулю, а при высокой поддержке несколько уступает Apriori. Последнее можно объяснить малым числом кандидатов и отсутствием необходимости оперировать со сложными древовидными структурами данных у Apriori.

Оба алгоритма являются линейно масштабируемыми по числу транзакций в БД. Но FP-дерево обладает лучшей масштабируемостью.

Исходя из сравнительного анализа можно сделать вывод, что алгоритм FP лучше чем Apriori по скорости работы и показателям масштабируемости, следовательно подходит для производственных и бизнес-приложений Data Mining, связанных с анализом больших объемов данных, например, в задаче выявления предпочтений покупателей супермаркета, в задачах оптимизации тарифной политики телефонных компаний.

В то же время алгоритм Apriori, в отличие от FP-дерева, не требует хранения в памяти массивных структур данных, поэтому может успешно использоваться для «небольших» задач, либо в тех областях, где стоит задача извлечения совсем «тонких» закономерностей, которым соответствуют высокие пороги support. Для такого рода задач Apriori может оказаться быстрее FP-tree из-за отсутствия накладных расходов, связанных с деревом.

Литература

1. J.Han, M.Kamber “Data Mining: Concepts and Techniques”, Morgan Kaufmann Publishers, 2000

2. R. Agrawal, R. Srikant "Fast Algorithms for Mining Association Rules", 1994, Proc. 20th Int. Conf. Very Large Data Bases, VLDB

3. J.Han, J.Pei, Y.Yin "Mining Frequent Patterns without Candidate Generation", 1999, 2000 ACM SIGMOD Intl. Conference on Management of Data

4. R.Srikant, R.Agrawal "Mining Generalized Association Rules", 1995, Future Generation Computer Systems

5. R.Srikant, R.Agrawal "Mining Quantitative Association Rules in Large Relational Tables", 1996, Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data

6. А. Шахиди "Введение в анализ ассоциативных правил", BaseGroup Labs

Петровский М.И., Сорокина Д.В.

Адаптация алгоритма классификации на основе нечетких деревьев решений для анализа многомерных данных[1]

Задача классификации

Входные данные:

- множество объектов S*
- K - количество классов
- конечный набор объектов $S_0 \subset S^*$ - тренировочный набор
- каждого объекта тренировочного набора известен его класс $\{i \mid i \in N, 1 \le i \le K\}$.

Требуется построить функцию $f : S^* \to \{1,2, \ldots ,K\}$, определяющую класс для каждого объекта из S*.

Чаще всего в качестве объектов классификации фигурируют векторы фиксированной длины. Во многих случаях используются и более сложные модели данных, включающие в себя вложенные таблицы, иерархии отношений атрибутов и так далее. В этой работе рассматривается модель данных на основе многомерного информационного куба [5]. В такой модели множество атрибутов объекта представляется в виде гиперкуба с фиксированным количеством измерений.

Многомерная модель данных

Интуитивно можно представить многомерный куб данных с метками (координатами) на ребрах куба. Каждая ячейка внутри куба находится на пересечении координат. Оси могут соответствовать таким параметрам, как, например, продукция, рынки, время, а внутри ячеек может храниться информация о продажах, соответствующим именно таким комбинациям времени, рынка и типа продукции, которые указывают координаты соответствующей ячейки.

[1] Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект № 03-01-00745

Основными понятиями в многомерной модели данных являются:

- измерение (Dimension)
- меры (Measure);.
- ячейка (Cell)

Значения, откладываемые на осях измерений, называются координатами. В многомерной модели данных координаты играют роль индексов, используемых для идентификации конкретных значений, находящихся в ячейках гиперкуба.

В свою очередь, ячейка - это поле, значения которого однозначно определяются фиксированным набором измерений. При использовании многомерной модели в задаче классификации и других задачах интеллектуального анализа данных ячейка выступает в роли атрибута. Тривиальный способ решения задачи – представить многомерный куб в виде вектора ячеек и применить алгоритм классификации, рассчитанный на работу с вектором. К сожалению, этот подход дает плохие результаты по причинам, о которых будет сказано ниже.

Структура многомерных данных – многомерный массив, определяется следующими параметрами: D - количество измерений и L_1, L_2, … L_D – длины массива по каждому измерению. Атрибут – позиция в этом массиве - определяется множеством координат $\{I_j \mid j = 1, \dots, D; 1 \le I_j \le L_j\}$. В одномерной структуре данных разным атрибутам могут соответствовать разные типы данных. В многомерной же структуре тип у всех атрибутов общий. Атрибут может быть определен на конкретном объекте q, тогда он принимет значение $q[I_1][I_2]\dots[I_D]$, или не определен ($q[I_1][I_2]\dots[I_D]$ не задано). Когда набор кубов таков, что на большинстве из них не определена значительная часть атрибутов, такой набор данных называется разреженным.

Высокая разреженность является особенностью многомерной модели данных, и именно она является основным камнем преткновения для попыток применять к таким данным стандартные алгоритмы интеллектуального анализа данных, рассчитанные на структуру в виде вектора. Любой алгоритм для работы с многомерными данными должен обязательно учитывать их разреженность.

Базовый алгоритм построения нечеткого дерева решений

В качестве классификатора будем рассматривать систему правил нечеткого вывода, а в качестве базового алгоритма построения классификатора - Soft Decision Tree[1], один из алгоритмов построения нечеткого дерева выбора для векторных данных [2,3].

Этот выбор сделан по следующим причинам:

- правила нечеткого вывода хорошо работают с действительными атрибутами;
- в сложных случаях, когда система не может однозначно классифицировать объект, она может выдать несколько классов с различными степенями принадлежности объекта к каждому классу;
- в алгоритм Soft Decision Tree включена генерация нечетких множеств, в отличие от других алгоритмов на основе нечеткой логики, где предполагается, что терм-множества заданы заранее.

Алгоритм Soft Decision Tree предназначен для решения задачи классификации для одномерных объектов.

Объекты, с которыми работает алгоритм - векторы действительных чисел длины N. Под атрибутом подразумевается позиция в векторе, а значение атрибута A на конкретном объекте q – число, находящееся в этой позиции данного вектора, q[A]. $C_j \subset S^*$ – множество объектов класса j.

На вход алгоритма подается S_0 - множество обучающих объектов, для которых известны их классы.

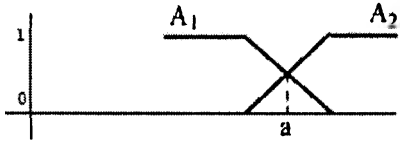
Алгоритм строит двоичное дерево, отвечающее следующим требованиям:

1. Каждому узлу дерева соответствует некоторое подмножество множества S_0: корню соответствует само S_0, остальным узлам некоторое подмножество множества, принадлежащего их предку.
2. Каждому нелистовому узлу соответствует некоторый атрибут A с областью определения U(A) и двумя нечеткими множествами A_1 и A_2, также определенными на U(A). Причем $\forall\, x \in U(A)$: $\mu_{A1}(x) + \mu_{A2}(x) = 1$. ($\mu_F(x)$ – функция принадлежности нечеткого множества F). Обозначим S - множество обучающих объектов, соответствующих этому узлу. Тогда первому потомку узла соответствуют объекты $q : q \in S,\ \mu_{A1}(\, q[A]\,) > \lambda$. ($\lambda$ - параметр алгоритма), второму – $q : q \in S,\ \mu_{A2}(\, q[A]\,) > \lambda$.
3. Каждому листу соответствует некоторый класс C_j.

Построение дерева.

Алгоритм начинает работу с корневого узла.
Для каждого атрибута выполнить шаги 1-5.

1. Обозначим: узлу соответствует множество S обучающих объектов q, $|S| = N$.
2. Сгенерировать G - множество точек–кандидатов на границу между нечеткими множествами, в SDT это точки, находящиеся посередине между любыми двумя соседними объектами разных классов.
3. Для каждого $a \in G$ построить разбиение U(A) двумя нечеткими множествами A_1 и A_2:



4. Посчитать выигрыш в энтропии для каждого разбиения.

$Gain = E_0 - E$, где

$E_0 = -\Sigma_j p_j \log p_j$
$p_j = |S_j| / |S|$, где $S_j = S \cap C_j$
$E = (N_1/N) E_1 + (N_2 / N) E_2$
$N = |S|$
$E_k = - \Sigma_j p_{jk} \log p_{jk}$, $k = 1,2$
$p_{jk} = N_{jk} / N_k$, $k = 1,2$
$N_k = \Sigma_i A_k(a_i)$, $k = 1,2$
$N_{jk} = \Sigma_{q_i \in S_j} A_k(a_i)$, $k=1,2$

5. Выбрать разбиение с наибольшим выигрышем и поставить его в соответствие выбранному атрибуту
6. Выбрать атрибут, разбиение по которому дает наибольший выигрыш по сравнению с разбиениями по другим атрибутам.
7. Построить два узла – потомка данного, каждый узел соответствует одному из двух нечетких множеств разбиения. На каждом потомке проверить критерии остановки построения дерева. Если они выполняются, соответствующий узел объявляется листом, и ему присваивается класс C_j, для которого p_{jk} максимально. Иначе этому узлу

ставится в соответствие множество обучающих объектов $\{q \mid q \in S, \mu_{A1}(q[A]) > \lambda\}$ и над ним выполняются те же действия, что и над родительским узлом.

После построения дерева алгоритм преобразует его в базу правил: каждому листу соответствует одно правило, построенное по следующей схеме:

Возьмем ветвь дерева от корня до выбранного листового узла. Пронумеруем эти узлы: корень обозначим D_0, его потомок – D_1, ..., лист - D_L. Обозначим A_{Di} – атрибут, по которому разбивается узел D_{i-1}, F_i – нечеткое множество, по принадлежности к которому определен набор обучающих объектов для узла D_i (множество из разбиения узла D_{i-1}).

Тогда соответствующее нечеткое правило выглядит так:

if A_{D1} is F_1 and A_{D2} is F_2 and ... and A_{DL} is F_L then q is C_j,

где Cj – класс, присвоенный листу.

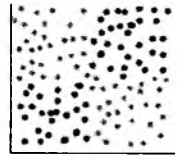
В других алгоритмах построения нечеткого дерева листу может соответствовать не один класс, а набор классов с коэффициентами, сумма коэффициентов по всем классам листа равна 1. Тогда правило будет выглядеть так:

if ... then q is C_1 with r_1, C_2 with r_2, ..., C_K with r_K.

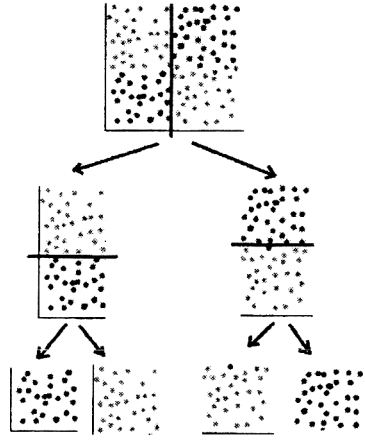
Недостатки обычных алгоритмов построения нечеткого дерева

Итак, в чем заключаются *недостатки алгоритмов, подобных SDT*, и почему они дают неудовлетворительный результат при применении к многомерным данным?

Все эти алгоритмы - жадные. На каждом шаге выбирается оптимальное разбиение только по одному атрибуту, поэтому некоторые эффективные решения могут быть упущены. Например, предположим, что в проекции на значения двух атрибутов множество обучающих объектов в текущем узле выглядит следующим образом:



Очевидно, что существует дерево разбиений, приводящее практически к стопроцентной классификации:



Но описанный алгоритм, скорее всего, не получит такого дерева, потому что не сможет корректно определить первое разбиение. В самом деле, для определения оптимального разбиения по одному атрибуту, SDT и подобные ему алгоритмы используют только информацию о распределении обучающих объектов в проекции на значения только этого атрибута. Но в указанной проекции объекты обоих классов имеют равномерное распределение. Следовательно, не может быть никакой уверенности в том, что предпочтение будет отдано нужному разбиению.

Подобные ситуации возникают, когда $S_0 \cap C_j$, множество обучающих объектов, принадлежащих одному классу, в пространстве, где каждое измерение соответствует одному атрибуту, является набором нескольких явно выраженных кластеров, либо явно невыпуклым множеством, что тоже можно представить как набор выпуклых кластеров. В этом случае эффективное разбиение S_0, скорее всего, будет разделять эти кластеры. Но при проекции на значения одного атрибута информация о кластерах в пределах одного класса часто бывает утеряна, и найти нужное разбиение этого атрибута тем самым становится практически невозможно.

Как отмечалось выше, многомерные данные обладают таким свойством, как разреженность, а именно это означает, что у

подавляющего большинства объектов, как обучающих, так и предназначенных для классифицирования, отсутствуют значения некоторых атрибутов. Алгоритмы, предназначенные для обработки одномерных данных, обычно игнорируют такие объекты. Очевидно, что при большой разреженности такой подход некорректен: могут оказаться забракованными почти все обучающие объекты. Отсюда следует, что алгоритм, предназначенный для обработки многомерных данных, должен использовать информацию о значениях атрибутов даже в тех случаях, когда объект данных из тренировочного набора определен не полностью.

При использовании для многомерных данных алгоритмов, предназначенных для обработки одномерных данных, многомерные данные приводятся к вектору, т.е. к одномерной структуре. Тем самым теряется важная информация о многомерной структуре, которая также может быть использована в многомерных алгоритмах. Возможные варианты использования многомерной структуры – использование значений агрегационной функции или выделение подмножеств среди множества атрибутов путем фиксирования некоторых координат.

Предлагаемый алгоритм

Нами разработан алгоритм построения классификационной модели для многомерных кубов, являющийся модификацией алгоритма SDT. Эта модификация учитывает многомерную структуру данных и может работать с разреженными кубами.

Используемые обозначения и терминология

Ячейку в многомерной модели мы будем называть простым атрибутом. Также в нашем алгоритме используется понятие сложного атрибута – это набор простых атрибутов с фиксированными значениями нескольких координат, другими словами, подмножество среза куба.

Срез многомерного куба определяется выбором чисел
$\{s_1, \ldots, s_m\} \subset \{1, \ldots D\}$, $m < D$
и соответствующих им значений координат
$\{Ks_j \mid j = 1, \ldots, m;\ 1 \le Ks_j \le Ls_j\}$
Срез куба - это множество простых атрибутов
$\{A[I_1][I_2]\ldots[I_D] \mid Is_j = Ks_j \ \forall\, j = 1,\ldots,m\}$

Какое именно подмножество среза куба является сложным атрибутом, определяется содержимым тренировочного набора, о чем будет подробно рассказано ниже.

Обучающий набор состоит из N D-мерных кубов. Мы рассматриваем такую структуру кубов, когда каждая клетка содержит либо действительное число, либо пустое множество (значение атрибута не определено).

Значение сложного атрибута – вектор значений входящих в него простых атрибутов. Значение сложного атрибута Ac на объекте q считается определенным, когда на q определены значения всех входящих в Ac простых атрибутов.

Описание алгоритма

Предлагаемый алгоритм строит нечеткое дерево, аналогичное дереву в SDT. В структуре этого дерева есть одно принципиальное дополнение – в разбиение каждого узла добавлен новый элемент, под названием «направление неопределенного атрибута». Каждое разбиение связано ровно с одним атрибутом, а это означает, что в условиях большой разреженности данных практически через любой узел будет проходить большое количество объектов, у которых этот атрибут не определен. Отправлять эти объекты в оба узла-потомка во многих случаях нецелесообразно, так как приводит к усложненной структуре конечного дерева без явной на то необходимости. Добавление возможности на стадии генерации дерева принять решения, какому потомку будут передаваться такие атрибуты, снимает эту проблему.

1. Построение Attr - набора классификационных атрибутов.

   - Вносим в Attr все простые атрибуты, и все возможные срезы куба в качестве сложных атрибутов.
   - для каждого простого атрибута
     - считаем кол-во объектов, в которых значение A определено – N_1
     - считаем кол-во объектов, в которых значение A не определено – N_2
     - считаем характеристику присутствия A: $X = N_1/(N_1+N_2)$
     - Если $X < \varepsilon_1$, удалить A из Attr и из всех сложных атрибутов, входящих в Attr

- Если в Attr появились одинаковые атрибуты, удалить двойников.
- для каждого сложного атрибута Ac
  - считаем кол-во объектов, в которых значение Ac определено – N_3 (множество хороших для Ac объектов)
  - считаем кол-во объектов, в которых значение Ac не определено – N_4 (множество плохих для Ac объектов)
  - считаем характеристику определенности Ac: $Y = N_3/(N_3+N_4)$
  - Если $Y < \varepsilon_2$, удалить A из Attr
- Удалить из Attr все сложные атрибуты, которые являются подмножествами других сложных атрибутов, входящих в Attr.

2. Построение оптимального разбиения узла.

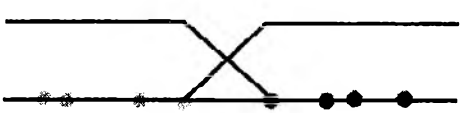
Для каждого атрибута находим оптимальное разбиение узла по нему.

Для простого атрибута:

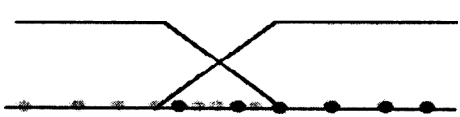
Взять множество значений атрибута на всех обучающих объектах, на которых этот атрибут определен. Построить множество разбиений и выбрать из них разбиение с наибольшим выигрышем в энтропии. Множество разбиений строится таким образом: на основе каждой пары классов, объекты которых присутствуют в узле, строим несколько разбиений по следующим критериям:

Если эти два класса не пересекаются и между ними находится еще хотя бы один класс, который не пересекается ни с одним из этих двух – не строим разбиения

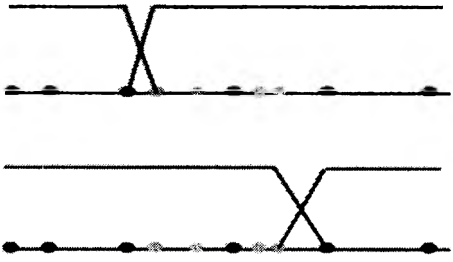
Если два класса не пересекаются, но между ними полностью не находится ни один другой класс – строим одно разбиение:



Если два класса пересекаются, но ни один не находится полностью внутри другого, тоже одно разбиение:



Если классы вложенные – два разбиения:



Разбиение на два узла происходит следующим образом: каждому узлу потомку соответствует одно из нечетких множеств разбиения и все те объекты, относящиеся к узлу-родителю, значение которых на данном множестве больше нуля.

После того, как построены нечеткие множества, определяется направление движения объектов с неопределенным атрибутом. Направление определяется голосованием. Для каждого класса определяется, куда лучше направлять его объекты, следующим образом:

Если определенные на атрибуте разбиения объекты этого класса, находящиеся в подмножестве тренировочного набора разбиваемого узла, перемещаются все в одном направлении, то вес этого класса в узле прибавляется к голосам, поданным за соответствующее направление.

Если часть определенных объектов перемещается в один узел, а часть в другой, то вес этого класса голосует за перемещение неопределенных объектов в оба направления.

Если ни один объект этого класса не определен на атрибуте разбиения, вес прибавляется и к голосам, поданным за левое направление, и к голосам, поданным за правое.

После этого значения, соответствующие трем возможным вариантам, сравниваются, и выбирается тот, который набрал максимальную сумму весов. В случае выбора левого или правого направления, объекты с неопределенным атрибутом перемещаются в соответствующий узел с тем коэффициентом k, который у них был в этом узле. В случае выбора двух направлений такие атрибуты перемещаются в оба узла-потомка с коэффициентом AND(k,0.5). Это касается как объектов тренировочного набора в процессе построения дерева, так и классифицируемых объектов в процессе классификации. Коэффициент AND(k, 0.5) гарантирует, что сумма коэффициентов объекта в левом и в правом потомках узла не превышает единицы, так же, как и при перемещении объектов, на которых атрибут разбиения определен.

Разбиение по сложному атрибуту – это поддерево с корнем в данном узле, которое строится следующим образом:

- Каждый хороший для данного сложного атрибута объект преобразовываем в вектор значений простых атрибутов, входящих в сложный.
- Для каждого класса:
  - выбираем из множества векторов вектора объектов, относящихся к данному классу.
  - Применяем к этим векторам алгоритм кластеризации X-Means[4].
- В результате мы получили расширенное множество классов (каждый старый превратился в несколько новых), причем все объекты обучающего набора из одного нового класса представляют собой выпуклое множество.
- Теперь по новому набору классов строим поддерево тем же методом, что и все дерево, но используя только разбиения по тем простым атрибутам, которые входят в сложный. Плохие объекты добавляем во все листья.
- Считаем энтропию полученного поддерева для первоначального набора классов.

После построения множества разбиений для различных атрибутов, как простых, так и сложных, сравниваем их энтропии. Находим разбиение с наибольшим выигрышем и добавляем его в дерево.

Если выбрано разбиение по сложному атрибуту, ветки соответствующего дерева нужно проверить на эффективность, и ненужные обрезать. Проверка на эффективность идет снизу вверх: берем узел, из которого исходят два листа, и считаем выигрыш в энтропии этого поддерева высотой один. Если он меньше некоторого порога, обрезаем эти два листа. И так до тех пор, пока все листовые разбиения малого поддерева не будут давать выигрыш больше минимального.

3. Проверяем критерии остановки. Обозначим K_q - коэффициент, с которым объект q пришел в этот узел из узла-предка. Объем узла V вычисляется как $\Sigma_{q \in S} K_q$. Объем класса j в данном узле $Vj = \Sigma_{q \in S \cap Cj} K_q$.

Критерии остановки:

1. Слишком мало объектов в узле. Соотношение V и общего количества обучающих объектов меньше α - заданного параметра алгоритма.

2. Слишком большой процент объектов, принадлежащих одному классу. $\exists j$: $Vj / V > \beta$

После остановки считаем коэффициенты присутствия каждого класса в полученном листе, равные V_j / V. Эти коэффициенты войдут в соответствующее листу правило.

Подсчет эффективности нечеткого дерева

Каждому листовому узлу ставим в соответствие нечеткое множество – пересечение всех множеств, соответствующих узлам на пути от данного листа к корню. Получаем M множеств B_i , M – количество узлов

Энтропия в корне (без учета разбиения):
$E_0 = -\Sigma_j p_j \log p_j$, где $p_j = [\Sigma_{q \in Cj} \Sigma_i B_i(q)] / [\Sigma_q \Sigma_i B_i(q)]$

Энтропия с учетом разбиения
$E = \Sigma_i (N_i / N) E_i$, где
$N_{ij} = \Sigma_{q \in Cj} B_i(q)$, $N_i = \Sigma_j N_{ij}$, $N = \Sigma_i N_i$ $(i = 1, \dots, M ; j = 1, \dots, K)$
$p_{ij} = N_{ij} / N_i$
$E_i = - \Sigma_j p_{ij} \log p_{ij}$

Выигрыш разбиения: $E_0 - E$

Тестирование

Верификация алгоритма производилась на двух тестах – на трехмерных и двумерных кубах. В рамках тестирования было проведено сравнение предложенного алгоритма с алгоритмом Microsoft Decision Tree, поставляемым вместе с SQL Server. Алгоритм Microsoft Decision Tree также строит классифицирующее дерево решений, но, в отличие от Soft Decision Tree и основанного на нем предложенного алгоритма, не использует нечеткую логику и генерирует строгие границы между узлами. По результатам тестирования Microsoft Decision

Tree показал худший результат по сравнению с предложенным алгоритмом.

К сожалению, на настоящий момент не существует стандартных наборов тестов для алгоритмов классификации многомерных данных, поэтому для оценки результатов были взяты два набора данных, не предназначенных специально для задачи классификации: данные о сделках и покупателях из базы FoodMart, поставляемой в качестве примера вместе с Microsoft Analysis Services [7], и реальные данные, соответствующие записям о продажах в небольшой коммерческой компании.

Для построения первого тестового набора мы взяли следующие измерения: время покупки (время года), тип товара и тип рекламы, которая привела покупателя. Один куб соответствует всем сделкам одного покупателя за год. В ячейках хранится общая сумма соответствующих покупок. Нужно классифицировать уровень дохода покупателя, отнести его к одной из четырех групп: 10-30 тысяч в год, 30-50, 50-70, 70 и больше. Тестовый набор содержит 3000 кубов, одна треть из которых использовалась как тренировочный набор.

Из-за высокой разреженности алгоритм Microsoft Decision Tree практически не смог работать на таком тестовом наборе, и правильно предсказал 29% случаев. В результате работы нашего алгоритма правильно были классифицированы 47% кубов.

Во втором тестовом наборе объект классификации представляет собой более простой куб с двумя измерениями. Одно измерение соответствует клиентам, другое – товарам. Первое измерение содержит 50 координат, второе 235. На пересечении находится сумма, на которую данный клиент купил данного товара. Каждый куб соответствует временному периоду, в течение которого операции проводил только один менеджер. Фамилия менеджера выбирается в качестве класса. Всего в базе фигурирует 8 фамилий, то есть в хранилище находятся кубы, принадлежащие восьми различным классам. Всего тестовый набор содержит 700 кубов. В качестве тренировочного набора случайным образом была выбрана четверть имеющихся данных.

В процессе тестирования на тестовом наборе Менеджер алгоритм Microsoft Decision Tree сделал верное предсказание в 55% случаев, в то время как предложенный алгоритм правильно определил класс у 78% кубов тестового набора.

Заключение

В рамках данной работы был разработан алгоритм классификации для многомерных кубов, содержащих вещественные меры и обладающих большой разреженностью. Алгоритм генерирует классификатор в виде нечеткого дерева решений, оперирующего понятием определенности атрибута на объекте. Для того чтобы исключить стандартные недостатки алгоритмов построения дерева решений, а именно неспособность обрабатывать классы сложной формы, введены дополнительная кластеризация внутри классов и построение поддерева по полученным кластерам. Поскольку для кластеризации и в последующем процессе построения поддерева требуется выбрать рабочий набор атрибутов, вводится понятие сложного атрибута – подмножества среза куба, из которого и выбирается требуемый набор.

В процессе тестирования было произведено сравнение с широко использующимся сейчас алгоритмом классификации, применимым к многомерной модели – Microsoft Decision Tree. Предложенный алгоритм показал более высокие результаты на обоих тестах.

Литература

1. Soft Discretization to Enhance the Continuous Decision Tree Induction. Yonghong Peng, Peter A. Flach. ECML/PKDD'01 workshop notes.

2. Continuous-Valued Attributes in Fuzzy Decision Trees. Jens Zeidler, Michael Schlosser. IPMU'96, Proceedings.

3. Generalization and Decision Tree Induction: Efficient Classification in Data Mining. Micheline Kamber, Lara Winstone, Wan Gong, Shan Cheng, Jiawei Han. RIDE'97, Proceedings.

4. X-Means: Extending K-means with Efficient Estimation of the Number of Clusters. Dan Pelleg, Andrew Moore. ICML 2000, Proceedings.

5. Tam, Y. J., Datacube: Its implementation and Application in OLAP Mining, Thesis submitted for the degree of Master of Science in the Department of Computer science of simon Fraser University, Canada, September 1998.

6. Microsoft Analysis Services

7. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/olapdmad/aggettingstart_80xj.asp

Раздел V
Инструментальные средства и сообщения

Леонов М.В., Новиков М.Д., Казакова Н.Л., Леонов В.М.

Программный инструментарий для интеграции и обработки библиографических данных в таксономических исследованиях.[1]

Введение

Библиографические данные - неотъемлемая составляющая большинства научных исследований. Особенно важную роль они выполняют в таксономии, где законность именования таксона определяется, в частности, приоритетом опубликования. Специфика предметных областей затрудняет разработку универсальных и удобных инструментов для работы с такими данными. Поэтому создание и сопровождение библиографических баз данных (БД) было и остается типичной задачей автоматизации научных исследований. В статье кратко представлен программный пакет, разработанный для вышеназванных целей в лаборатории вычислительного практикума и информационных систем факультета ВМиК МГУ для коллег из Ботанического сада МГУ.

1. Предыстория и актуальность задачи.

При автоматизации предметной области весьма распространен подход, получивший образное название «архипелага автоматизации» [1], суть которого в той же работе поясняется фразой «дорожки сначала протаптывают, а затем асфальтируют». В нашем случае в целом оправданное применение этого подхода привело к параллельному сосуществованию нескольких таксономических БД, каждая со своим библиографическим блоком. Заметим, что существуют различные стандарты библиографических ссылок – есть стандарт для так

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

называемых первоописаний таксонов, есть ряд стандартов для ссылок в наиболее уважаемых международных издательствах и редакциях. Чтобы дать более полное представление об объемах данных в этой области, приведу один пример. В 70-х годах прошлого века начал выходить многотомный труд под названием «Таксономическая литература» (каждый том с более 1000 страницами мелкого текста) с данными об авторах, журналах, сборниках и монографиях таксономической ботаники, и каноническими правилами сокращенного написания основных источников. Так как не существует единой библиографической БД, из которой исследователь мог бы извлекать необходимый ему «срез» данных, стандартная ситуация в небольшом научном коллективе такова. Исследователь, занимающийся, к примеру кариологией, создает свою БД с кариологическими данными по своей таксономической группе, в которой присутствует, конечно, и библиографический блок, причем ему, как правило, некогда вводить полные данные по источникам, либо они ему просто недоступны - и в результате он накапливает эти данные в той форме, в какой ему необходимы для текущих публикаций и сводок. Через несколько лет, когда кариологическая БД в основном завершена, тот же исследователь (или его коллега) сосредотачивает свои усилия на другом аспекте исследования таксонов, и создает новую БД, в которой присутствует свой библиографический блок. Естественно, рано или поздно ему приходится наводить порядок - согласовывать свои данные, да и требования издательств (западных) повышаются. Например, есть стандарт, требующий для работы, написанной не на латинице, кроме перевода названия на английский язык, также и транслитерацию оригинального названия латинскими буквами. Появились также источники в Интернет - электронные библиографические каталоги – со своими форматами, которые надо учитывать.

Таким образом, возникает задача интеграции различных библиографических БД и других источников, согласования данных, и обеспечения возможности репликации из интегрированной БД «исправленной и обогащенной» частной БД. Важной задачей является также подготовка к печати библиографического списка в одном из требуемых форматов. Именно для решения этих задач нами был разработан (и продолжает развиваться) пакет для создания и обслуживания интегрированной БД под названием BIBLIUM.

2. Суть подхода и состав пакета.

Общая схема работы пакета представлена на рис. 1. Библиографические файлы в наших таксономических БД ASIUM, GNOM и CARUM (см. [2]) конвертируются с помощью специальной программы в формат интегрированной БД BIBLIUM, в котором

предусмотрены все известные к данному моменту пожелания пользователей, включая указания ключевых слов. Так как все эти БД имеют формат dbf, то программу-конвертер решено было написать на языке FoxPro. Несмотря на бытующее мнение об архаичности СУБД FoxPro, ее весьма успешно продолжают использовать для управления локальными БД. Интерфейс этой же программы позволяет редактировать введенные данные, фильтровать их по различным критериям (по времени издания, списку и лексикографическому интервалу фамилий авторов, ключевым словам). Интегрированная БД содержит записи трех типов, соответствующих статьям в журналах, статьям в сборниках и монографиям. То есть фактически была реализована ИПС с возможностями пакетного и интерактивного ввода библиографических данных.

Вторая программная компонента пакета – программа генерации нужного фрагмента библиографических данных в формате XML, причем с возможностью отбора полей исходной БД и переименования тэгов XML-документа. В этом случае, как и в работе [3], XML используется как универсальный промежуточный формат интегрированных данных.

На третьем этапе обработки данных происходит визуализация сгенерированной сводки с помощью Internet Explorer'а и одной из XSL-таблиц, соответствующей требуемому формату вывода. При необходимости полученную сводку можно записать в файл в формате HTML.

Кроме перечисленных компонент, в пакет входят программные агенты на языке Perl для автоматического опроса по заданному списку важнейших библиографических Интернет-каталогов по таксономической ботанике [4]. Такой опрос существенно экономит время, но полученные таким образом данные нельзя автоматически вносить в интегрированную БД, они подлежат кропотливой проверке специалистами.

Пакет легко расширяем: если возникнет необходимость интегрировать библиографические данные из нового источника, достаточно написать дополнительную утилиту, воспользовавшись готовыми функциями контроля вводимых данных.

После завершения опытной эксплуатации БД BIBLIUM, содержащей в себе основную библиографию по таксономии растений семейства Зонтичных, ее предполагается сделать доступной через Интернет на совместном сайте факультета ВМиК и Ботанического сада МГУ.

Заключение

Необходимость интеграции БД с различными форматами данных возникает в настоящее время в самых разных областях и масштабах, отражаясь, например, в многочисленных проектах реинжиниринга информационных систем. В некоторых случаях интеграции стараются избежать, предпочитая этому централизованному подходу к извлечению данных из различных источников так называемый «федеративный подход». В нашем случае, когда речь идет об однородных по сути, отличающихся лишь структурой и формой записи, библиографических данных, наш подход представляется наиболее адекватным и применимым при решении задач этого класса.

Литература.

1. Громов Г.Р. Очерки информационной технологии. - М.: ИнфоАрт, 1993 г. - 336 с.

2. Леонов М.В. Комплекс программ для автоматизации исследований в таксономической ботанике. / Сб. "Программные системы и инструменты" №2, - М., 2001, стр.168-172.

3. Леонов М.В., Мошкин К.Б., Леонов В.М. XML как средство модернизации унаследованных баз данных. / Сб. "Программные системы и инструменты" №3, - М., 2002, стр.28-32.

4. М.В.Леонов, Д.Ю.Малинин. Автоматизированный мониторинг и накопление электронных данных по растениям семейства Umbelliferae из источников Интернет. В кн.: Информационные и телекоммуникационные ресурсы в Зоологии и Ботанике. – Санкт-Петербург, 2001 г., стр. 99-100.

Брусенцов Н.П., Владимирова Ю.С.

Логико-алгебраический процессор

Логико-алгебраический процессор представляет собой программно реализованную на компьютере систему обработки не количественных (числовых), а качественных (логических) данных. Компьютер служит в ней не столько вычислителем в арифметическом смысле, сколько преобразователем выражений алгебры логики, отображающих взаимосвязи качеств и охарактеризованных посредством них вещей. Зафиксировав избранные взаимовсязи в виде посылок, имеем логические уравнения, решением которых получаются вытекающие из данных посылок умозаключения. Таким образом, компютеризируется именно тот способ математической логики, на котором настаивали Лейбниц, а затем Буль – дедукция сведена к решению уравнений, осуществляемому теперь компьютером.

Как средство реализации алгоритмов компьютер в составе логического процессора сохраняется неизменным. Специализация обеспечивается введением типов данных, представляющих стандартные формы выражений алгебры логики. Эти типы определяются в диалоговой системе структурированного программирования ДССП как *конструкты* – процедуры, обладающие собственной памятью установленного формата и параметрически задаваемой емкости, интерпретация содержимого (значения) которой осуществляется базисным набором операций соответствующего типа [1].

Проще говоря, формы алгебраических выражений кодируются векторами битов (либо тритов) подобно тому, как такими же векторами кодируются числа, но операции теперь не арифметические, а логические, и код обретает иной смысл, который в самом общем выражении состоит в следующем.

Имеется n попарно различных критериев (качеств, признаков), относительно которых охарактеризованны рассматриваемые вещи. Совокупность этих n критериев, называемых далее первичными *терминами* составляет *Универсум* – Вселенную рассмотрения. Если вещи охарактеризованны по каждому из n критериев однозначно: либо критерию x данная вещь необходимо удовлетворяет (не может не удовлетворять), либо антиудовлетворяет, то вещам сопоставлены *четкие* подсовокупности n-терминной совокупности. Полная совокупность таких вещей-индивидов есть булеан совокупности терминов - всего в Универсуме 2^n индивидов.

В булевой алгебре четкие подсовокупности терминов выражаются n-арными элементарными (индивидными) конъюнкциями, в которые термины, присущие вещи, входят непосредственно, а

антиприсущие – инвертированными, помеченными штрихом. Например, при n=4 конъюнкция $xy'zw'$ означает индивид, которому присущи x и z, а y и w – антиприсущи.

Четкая подсовокупность n-терминной совокупности естественно кодируется вектором, биты которого однозначно сопоставлены терминам (проиндексированы терминами). Если термин принадлежит подсовокупности, то его биту присваивается значение 1, а если не принадлежит, то 0. Так, конъюнкция $xy'zw'$ кодируется вектором 1010. Но надо указывать тип кодируемой совокупности, поскольку она может быть конъюнктивной либо дизъюнктивной. Индивидная конъюнкция отображается конструктом *n-битная К-шкала*, предполная дизъюнкция – *n-битная Д-шкала.*

Произвольная n-арная булева функция представима в совершенной дизъюнктивной нормальной форме (СДНФ) *2^n-битной ДК-шкалой*, биты которой сопоставлены значениям n-битной К-шкалы, т. е. индивидным конъюнкциям. Совершенная конъюнктивная нормальная форма (СКНФ) аналогично кодируется *2^n-битной КД-шкалой*, отображающей конъюнктивные подсовокупности n-битных Д-шкал (предполных дизъюнкций терминов).

Замечательно, что операции булева отрицания, конъюнкции и дизъюнкции n-арных выражений выполняются как теоретико-множественные инверсия, пересечение и объединение подсовокупностей индивидных конъюнкций либо предполных дизъюнкций, т. е. как побитные отрицание, конъюнкция и дизъюнкция соответственно ДК- и КД-шкал, несложно реализуемые на современных компьютерах. В сочетании с операцией, формирующей двойственное выражение путем инвертирования типа шкалы, эти средства позволили осуществить пакет практически полезных алгебраических процедур, в частности, выявление отношений между данными выражениями, минимизацию выражений, решение булевых уравнений [2].

Для представления сокращенных и минимальных ДНФ и КНФ, которые необходимо содержат неиндивидные конъюнкции и непредполные дизъюнкции, являющиеся нечеткими совокупностями терминов используется конструкт типа цепь n-тритных слов соответственно К- и Д-шкал [3].

Реализованный в ДССП логико-алгебраический процессор наследует стековую архитектуру и диалоговый характер этой системы программирования: операции задаются в префиксной бесскобочной записи, процессор выдает сообщения, информирующие пользователя о готовности к работе, и запросы вида задачи, параметров и выражений, составляющих ее условие. Первым следует запрос количества и упорядоченного перечня терминов, составляющих Универсум, затем запрашивается вид задачи, составляющие ее выражения и уравнения, а

также "искомые" термин либо выражение. Результаты выдаются с необходимыми текстовыми комментариями.

Булевы выражения вводятся в общепринятой алгебраической записи в виде строк, автоматически отображаемых процессором в ДК-шкалы. При выдаче результатов производится обратное преобразование шкал и цепей в строки.

Литература

1. 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. С. 16-43.
2. Владимирова Ю.С. Конструктная реализация булевой алгебры // Интегрированная система обучения, конструирования программ и разработки дидактических материалов. – М.: Изд-во ф-та ВМиК МГУ, 1996. С. 44-69.
3. Брусенцов Н.П., Владимирова Ю.С. Троичная компьютеризация булевой алгебры // Цифровая обработка информации и управление в чрезвычайных ситуациях. - Минск: Институт технической кибернетики НАНБ, 2002. Том 2. С. 195-199.

Бурцев А.А., Рамиль А.Х.

Средства объектно-ориентированного программирования в ДССП

Диалоговая Система Структурированного Программирования (ДССП) [1,2] задумывалась как инструментарий, призванный облегчить разработку эффективных программ для мини- и микрокомпьютеров. Она создавалась для поддержки технологии структурированной разработки программ. Но подобно другим языкам (Си, Паскаль, Forth) ДССП постоянно развивалась и совершенствовалась, чтобы отвечать потребностям современных технологий программирования.

При построении новой версии ДССП [3] (ДССП/С), ядро которой в целях обеспечения мобильности системы разрабатывалось на языке Си, в качестве одной из задач была выполнена такая модификация механизма работы с данными, которая позволила в дальнейшем легко расширять арсенал типов данных, понимаемых системой, а также дополнять ассортимент универсальных префиксных операций над переменными. Впоследствии это позволило создать для ДССП средства наращивания словаря системы новыми словами, с помощью которых можно не только объявлять величины новых типов, но также определять и новые слова, задающие операции над величинами этих типов. В результате в ДССП были обеспечены возможности объявления новых типов данных сначала как структур (записей в Паскале), а затем и как классов, т.е. обеспечены средства объектно-ориентированного программирования (ООП).

Далее рассматриваются некоторые аспекты модификации ДССП на пути реализации для нее ООП-средств.

1. Модификация механизма ДССП для работы с данными

В версии ДССП/С интерпретатор сшитого кода, образующий сердцевину ядра системы, в целях его переносимости потребовалось разработать на языке Си . Для этого пришлось существенно изменить структуру сшитого кода системы, т.е. переделать внутреннее строение тел всех слов-процедур, а также дескрипторов типов и тел слов, представляющих данные (константы, переменные, массивы). В результате такой модификации в механизм работы с данными были внесены существенные изменения.

Во-первых, в дескрипторах типов таблица ссылок на процедуры, реализующие префиксные операции над переменными,

теперь растет в сторону увеличения адресов, что позволяет легко ее дополнять. Во-вторых, сами эти процедуры теперь стало возможным определять не только как слова-примитивы (в машинном коде), но и как обычные слова-процедуры (через:). Для обеспечения в этих процедурах доступа к телу обрабатываемой переменной и дескриптору соответствующего ей типа в ядре системы предусмотрены две системные переменные-указатели: DBAdr и TypeAdr. Еще одна системная переменная DTYPE используется на стадии выполнения компилирующих слов (таких, как VAR , VCTR , ARR и др.), определяющих тела слов для объявляемых переменных, векторов и многомерных массивов. Она содержит ссылку на дескриптор типа, которая будет использована при очередном объявлении переменной и будет заноситься в соответствующее поле тела образуемой переменной. Значение же этой переменной должно устанавливаться теми словами (например, LONG), которые предусматриваются для определения типа переменной перед ее объявлением словом VAR.

Для добавления нового типа данных к представленному механизму работы с переменными требуется: 1) создать процедуры-исполнители, реализующие стандартные операции доступа (‘, !, !+, !- и т.д.) к величинам нового типа; 2) сформировать дескриптор нового типа, расставив в нем ссылки на эти процедуры-исполнители; и 3) сформировать тело и добавить в словарь новое слово, вызов которого обеспечит установку в переменную DTYPE значения ссылки на дескриптор этого нового типа. Если же для работы с величинами нового типа желательно еще и расширить набор префиксных операций, добавив новые операции, то для них потребуется еще 4) определить тела новых слов, запускающих механизм выполнения этих префиксных операций в зависимости от типа переменной, стоящей следом по телу сшитого кода.

В результате проделанной модификации механизма работы с данными, во-первых, была обеспечена возможность создавать в ДССП новые типы данных на языке самой системы. А во-вторых, открылась возможность автоматизировать весь комплекс действий (1- 4), направленный на построение новых типов данных вместе с новыми операциями для них.

2. Средства построения новых типов данных как записей

Сначала были созданы средства построения в ДССП новых типов данных подобно записям (Паскаля) или структурам (языка Си). Общий вид объявления в ДССП нового типа данных как структуры выглядит следующим образом:

STRUCT: <имя_структуры> [имя типа новой структуры]
VAR *<имя_поля>* **... VAR** *<имя_поля>* [поля структуры]
;STRUCT [Вместо VAR можно использовать VCTR, ARR]

Объявление новой структуры в ДССП начинается командой **STRUCT:**, за которой задается имя типа объявляемой структуры, и заканчивается командой **;STRUCT** . Все команды объявления данных (с помощью служебных слов VAR VCTR и им подобных), расположенные в промежутке между командами **STRUCT:** и **;STRUCT**, определяют не самостоятельные переменные, а поля формируемой структуры.
Пример объявления типа для работы с величинами-датами:

```
STRUCT: TDATE [ тип для представления даты ]
        VAR .Year BYTE VAR .Month BYTE VAR .Day ;STRUCT
```

В результате такого объявления формируется не только дескриптор нового типа (TDATE), но и тела слов - полей (.Year, .Month, .Day), составляющих новую структуру. В отличие от обычных переменных в телах таких слов задается ссылка на функцию вычисления адреса переменной относительно начала размещения структуры.
После объявления типа TDATE можно объявлять структурные переменные такого типа, а также строить из них массивы:

```
TDATE VAR Date  TDATE VAR NewDate  TDATE 9 VCTR Dates
```

Над структурными переменными целиком можно выполнять операции копирования одной структуры в другую (!, !!!), обнуления всех полей структуры (!0), а также получения размера (SIZE?) структуры (ее длины в байтах) и ее размерности (DIM?). При вызове имени структурной переменной в стек посылается адрес ее размещения (то же самое делают и префиксные операции ', !1, !1+, !1-, !+, !-).
Примеры:

```
Date 5 ! Dates [ копирование даты целиком Dates(5):=Date ]
Date !!! Dates [ Dates(i):=Date для i=0,1,..,9 ]
```

Кроме того, над отдельными полями структуры можно выполнять такие же стандартные операции, как с обычными переменными, но перед каждой такой операцией требуется заслать в стек адрес структурной переменной, к полю которой будет применяться эта операция. Примеры:

Date ['Date] **.Year** [Date.Year]
5 Dates ['Dates(5)] **! .Year** [Dates(5).Year:= Date.Year]

3. Объявление новых типов как классов

Для объявления в ДССП нового типа данных как класса используется синтаксическая конструкция, аналогичная объявлению структуры, только внутри класса (между служебными словами **CLASS:** и **;CLASS**) разрешается объявлять не только поля, но еще и методы:

CLASS: *<имя_класса>* [имя типа нового класса]
VAR *<имя_поля>* ... **VAR** *<имя_поля>* [объявление его полей и]
METHOD <*имя_метода*> ... METHOD <*имя_метода*> **[методов]**
;CLASS

Причем объявления полей и методов можно чередовать в любом порядке, а также предварять их словом::, если требуется оставить их имена в подсловаре даже после его чистки (CLEAR). Перед VAR можно задать тип объявляемых полей. Кроме VAR можно также использовать и слова для объявления массивов (VCTR ARR). Вместо слова **METHOD** можно использовать его вариации для назначения нового имени уже известному методу:

METHOD= *<прежнее_имя_метода>* *<новое_имя_метода>*
[N] **METHOD#** *<новое_имя_метода>* [N – номер прежнего метода]

Для назначения процедуры, реализующей объявленный метод класса, предлагаются две синтаксические конструкции:

<имя_класса>:**M:** *<имя_метода>* *<тело_реализации_метода>* **;**
<имя_класса> :**M=** *<имя_метода>* *<имя_процедуры>*

В первой из них задается само тело процедуры, а во второй указывается лишь имя процедуры в предположении, что такая процедура уже была предварительно определена:

: *<имя_процедуры>* ['X] ...*<тело_ реализации_ метода>*... [ ] ;

Для выполнения действий со всем объектом целиком могут применяться те же операции, что и со структурой. В частности, вызов слова по имени объекта будет засылать в стек адрес размещения этого объекта в памяти. А с полями объекта разрешается выполнять такие же действия, как и с полями структуры. Для вызова же метода объекта используется синтаксическая конструкция, аналогичная вызову префиксной операции над переменной:

[ParamsBefore] *<имя_метода>* *<имя_объекта>* [ParamsAfter]

А имя объекта должно быть предварительно объявлено с помощью конструкции вида:

<имя_класса> **VAR** *<имя_объекта>*

Перед вызовом объекта, как и перед вызовом всякой процедуры, можно приготовить в стеке величины [ParamsBefore], необходимые для исполнения метода (как входные параметры), а после его вызова - использовать величины, оставленные им в стеке в результате своего выполнения [ParamsAfter] (как выходные параметры). При этом следует помнить, что в момент вызова процедуры-исполнителя метода в стек добавится еще и адрес объекта, к которому данный метод применяется.

Приведем пример объявления класса QUEUE для работы с очередью:

```
:: CLASS: QUEUE [ Класс ОЧЕРЕДЬ ]
  [--- поля данных, составляющих очередь ---]
   VAR .cnt [ кол-во элементов в очереди ]
   VAR .n  [ индекс-указатель начала очереди в массиве ]
   VAR .k  [ индекс-указатель конца очереди в массиве ]
 MaxLen 1- LONG VCTR .M [массив для хранения элементов очереди]
    [--- методы , представляющие операции над очередью ---]
:: METHOD Init [ инициировать, начать работу очереди ]
:: METHOD Show [ показать на экране состояние очереди ]
:: METHOD Get  [ взять элемент из очереди на вершину стека ]
:: METHOD Put  [поместить элемент из вершины стека в очередь]

:: METHOD Empty? [ проверить, пуста ли очередь ]
:: METHOD Full?  [ проверить, полна ли очередь ]
:: METHOD Done [ завершить, закончить работу очереди ]
:: METHOD= ! =>  [  новое имя метода для копирования очереди ]
;CLASS
```

Примеры реализации некоторых методов этого класса:

```
QUEUE:M: Init [Q] C !0 .cnt C !0 .k [Q] !1 .n  [ ] ;
QUEUE:M: Put [x,Q] C .Full? IF+ Full! C .k +1mod C C3 ! .k
        [x,Q,k] C2 !1+ .cnt E2 [x,k,Q] ! .M [ ] ;
QUEUE:M: Get [Q] C .Empty? IF+ Empty! C .n C2 .M E2
          C !1- .cnt C .n +1mod E2 [y,n+1,Q] ! .n [y] ;
    : .Empty? [Q] .cnt 0 <= [1/0] ;
```

```
QUEUE:M= Empty? .Empty?
```

Примеры объявлений объектов типа QUEUE:

```
  QUEUE VAR Q   [ Q - отдельный экземпляр класса QUEUE ]
9 QUEUE VCTR VQ  [ массив очередей VQ(0:9) ]
```

Примеры выполнения ряда действий с объявленными объектами:

```
Q [адрес Q] 3 VQ ['VQ(3)] [вызов операций по имени объектов]
Q .cnt [Q.cnt] [получение значения кол-ва элементов очереди Q]
Q !0 .cnt [ Q.cnt:=0 обнуление счетчика элементов очереди Q ]
3 VQ !  Q  [ Q:= VQ(3) копирование очереди целиком ]
3 VQ => Q  [ то же, но применяя новое обозначение метода ]
Init Q [ вызов метода для инициализации очереди Q ]
77 Put Q [ помещение в очередь Q числа 77 ]
3 Get VQ [y][ взятие значения (y) из очереди VQ(3) ]
```

4. Объявление класса-наследника

Новый класс в ДССП можно объявить на основе уже имеющегося класса путем наследования его свойств (полей и методов), и дополняя при этом его определение новыми полями и методами. Для этого сразу после объявления имени нового класса следует применить служебное слово **SUBCLASS**, после которого задать имя наследуемого класса. С учетом наследования синтаксическая конструкция для объявления нового класса будет выглядеть так:

```
CLASS:  <имя_нового_класса>
 SUBCLASS  <имя_наследуемого_класса>
 VAR <имя_поля> ... VAR <имя_поля>  [объявления новых полей и ]
  METHOD <имя_метода> ...  METHOD <имя_метода> [ методов ]
;CLASS
```

В качестве примера объявим класс DEQ как наследник класса QUEUE и продемонстрируем приемы выполнения действий с объектами типа DEQ:

```
:: CLASS:  DEQ  SUBCLASS QUEUE [ наследует класс QUEUE ]
  :: METHOD Put_  [поместить элемент в начало очереди]
  :: METHOD Get_  [ взять элемент с конца очереди ]
  :: METHOD Count [ узнать количество элементов в очереди ]
;CLASS [--- Далее реализация методов класса DEQ: ---]
```

```
DEQ:M: Put_ [x,Q] C .Full? IF+ Full!
        C .n -1mod C C3 ! .n  C2 !1+ .cnt  E2 ! .M [ ] ;
DEQ:M: Get_ [Q] C .Empty? IF+ Empty!
        C .k C2 .M E2 C !1- .cnt C .k -1mod E2 ! .k [y] ;
DEQ:M= Count .cnt
DEQ:M: Show [Q] ."Double-End-"  Show AS QUEUE [] ;
[--- Примеры выполнения действий с объектами-наследниками: ---]
DEQ VAR DQ [ дек, который может использоваться и как очередь ]
  DQ => Q [ Q:= DQ копирование дека как очереди ]
  Init  DQ [ вызов метода для инициализации дека ]
  DQ .cnt [получение значения кол-ва элементов в деке]
 x Put  DQ [ помещение в конец дека значения x ]
  Get  DQ [y][ взятие значения (y) из начала дека ]
   x Put_ DQ [ помещение в начало дека значения x ]
  Get_ DQ [y][ взятие значения (y) из конца дека ]
   Show DQ [ распечатать содержимое дека на экране]
```

5. Механизм действия методов класса

Механизм действия методов класса (см. рис. 1) построен на основе механизма исполнения префиксных операций над переменными и является его усовершенствованным вариантом. При объявлении нового класса дескриптор для него создается путем первоначального копирования дескриптора либо наследуемого им класса, либо стандартного базового класса, если наследуемый класс не был объявлен. После чего к дескриптору добавляются ссылки на тела исполнителей новых методов, а также обновляются в нем ссылки для перекрываемых методов. Дескриптор базового класса содержит ссылки на процедуры-исполнители 12-ти стандартных префиксных операций (теперь методов: ‘, !, !+, !- и т.д.), которые для всех объявляемых классов будут реализовывать эти операции так же, как они действуют для структур.

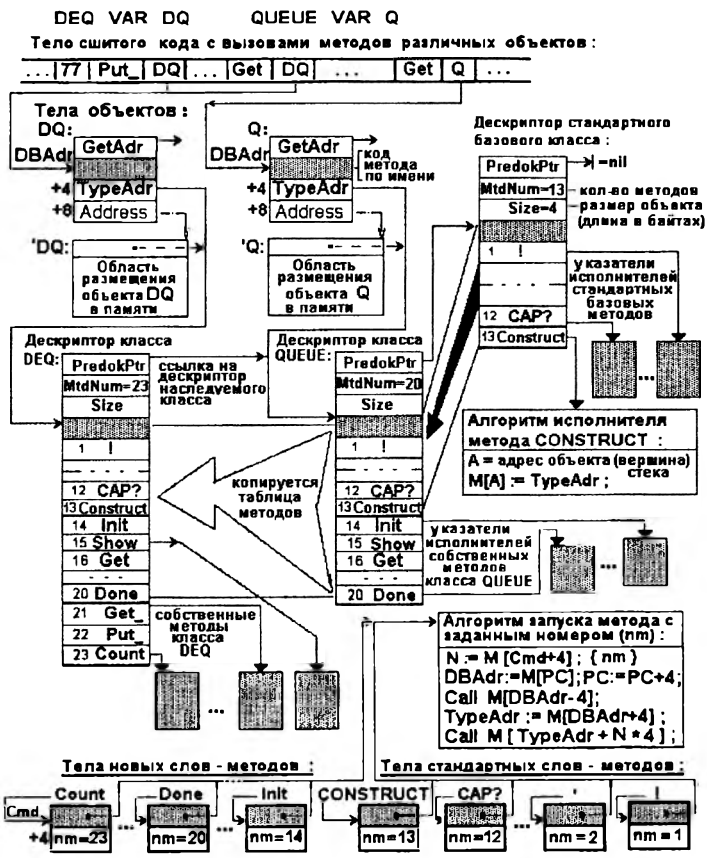


рис. 1.

Механизм действия методов класса.

6. Дополнительные возможности механизма классов

В общем случае методы должны применяться к объектам подобно префиксным операциям, при этом имя объекта должно следовать сразу же за именем применяемого метода. Но в ряде случаев оказывается необходимым применять к объекту одного класса такие методы, которые разрешены для другого класса, или просто работать с произвольным участком памяти как с объектом нужного типа. Для реализации этой возможности механизм классов дополнен служебным словом **AS**:

[адрес_объекта] **<имя_метода> AS <имя_класса>**

При префиксном вызове метода, когда имя объекта задается вслед за методом, никаких проблем с обеспечением свойства полиморфизма объектов не возникает, так как с именем объекта всегда связана ссылка на дескриптор объекта, из которого, в свою очередь, можно получить и адрес размещения объекта в памяти, и ссылку на дескриптор его класса, где расположена таблица, содержащая ссылки на процедуры-исполнители методов. Если же адрес объекта задается в стеке, то тип объекта приходится задавать особо (после **AS**). Но можно и сохранить ссылку на тип в памяти объекта при его инициализации. Для этого предназначен метод **CONSTRUCT**, вслед за которым необходимо задать имя объекта.

Сконструированный таким образом объект теперь может сам идентифицировать свою принадлежность определенному классу. Значит, к нему можно применять методы, указывая лишь адрес его размещения в памяти. Для вызова метода по отношению к такому самоидентифицирующемуся объекту предлагается служебное слово **SELF**, которое (как и слово **AS**) должно следовать за именем метода вместо имени объекта:

[адрес_объекта] **<имя_метода> SELF**

Описанные средства ООП были опробованы при создании библиотеки классов простых графических фигур, способных перемещаться по экрану под управлением клавиатуры. Создание подобной библиотеки позволило в ходе реальной работы критически оценить предлагаемые для ДССП ООП-средства и частично усовершенствовать их. Полученный опыт объектно-ориентированной разработки в ДССП позволяет заключить, что, в целом, предложенные ООП-средства обеспечивают построение в ДССП полноценных объектов, обладающих свойствами инкапсуляции, наследования и полиморфизма.

Литература

1. Брусенцов Н.П., Захаров В.Б., Руднев И.А., Сидоров С.А. Диалоговая система структурированного программирования ДССП. // Диалоговые микрокомпьютерные системы. - М.: Изд-во Моск. ун-та, 1986, с. 3-21.

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

3. Бурцев А.А., Франтов Д.В., Шумаков М.Н. Разработка интерпретатора сшитого кода на языке Си. // Вопросы кибернетики. Сб. статей под ред. В.Б. Бетелина. - М., 1999, с.64-76.

Маслов С.П., Рамиль А.Х., Сидоров С.А.

Реализация МСО "Наставник" на микрокалькуляторе МК-85

Обосновывается перспективность развития системы обучения "Наставник" [1] с ориентацией на использование автономных, т.е. не нуждающихся в постоянной связи с центральным компьютером, терминалов. Обсуждается специфика и способы реализации терминала путем программной модификации существующих тиражируемых устройств. Приводятся краткие сведения об отечественном калькуляторе "Электроника МК-85" и его штатном ПО. Описываются процесс формирования модифицированного ПО и используемые для этого средства.

Введение

Версия "Наставника", описание которой является предметом этой статьи, юбилейная - в 2002 году системе исполнилось 30 лет. Столь долгая жизнь и востребованность в условиях быстрой смены поколений аппаратуры и программного оснащения без всякой натяжки могут считаться феноменальными. В основе феномена лежит заложенная в "Наставник" и полностью оправдавшая себя на деле реалистическая концепция использования компьютеров для обучения, наличие учебных и методических материалов, оперативный перенос системы на разные компьютерные платформы и своевременное совершенствование ее аппаратуры. Вместе с тем, в эволюции "Наставника" имели место и негативные моменты - несмотря на благоприятное к ней отношение со стороны многих потенциальных потребителей, система не получила широкого распространения. Продолжительный опыт разработки и внедрения многочисленных версий "Наставника" дает основание назвать две главные причины этого: уникальность аппаратуры системы и использование проводов для связи управляющего компьютера с терминалами.

В большинстве версий системы [2] применялись специально разработанные (уникальные) узлы. Это было вызвано тем, что в ассортименте готовых изделий не удавалось найти ничего подходящего. Уникальные устройства (в первую очередь терминалы) проектировались с расчетом на доступность широкому кругу потребителей – школам, ПТУ, ВУЗам и т.д. Однако найти желающих делать простые и дешевые устройства удавалось с трудом. Даже в тех

случаях, когда выпуск уникальных узлов был организован, существовала причина, затрудняющая установку системы для тех, кто смог их приобрести. Она заключалась в наличии проводов между компьютером и терминалами - разводки. Практика показала, что самостоятельно сделать разводку оказывалось не по силам большинству желающих иметь "Наставник". Разводка - "ручная работа", а она простой и дешевой не бывает.

Выходом из положения был бы отказ от проводов вообще и организация дела таким образом, что деятельность обучаемого (хотя бы на протяжении занятия) осуществляется вообще без вмешательства центрального компьютера. Терминал при этом самостоятельно управляет обучаемым, а связь с компьютером (при необходимости) происходит вне занятия: до него (загрузка управляющей информации) и после (выгрузка протокола проведенного занятия) через стандартный связной порт.

Такой автономный терминал на нынешнем уровне цифровой техники вполне реализуем на микропроцессоре. Может показаться, что при этом будет устранена лишь одна из причин малой распространенности системы - отпадет необходимость в разводке, а сам терминал по-прежнему останется уникальным. Это не так. В последнее десятилетие появились и получили широкое распространение недорогие устройства типа электронных органайзеров, записных книжек, переводчиков, калькуляторов, телефонных аппаратов – "ширпотребе". Эти устройства выполнены на микропроцессорах и, как правило, могут быть модифицированы таким образом, что имеющийся ассортимент функций дополнится возможностью работы в качестве автономного терминала "Наставника". Модификация заключается в пополнении штатного ПО. Само устройство не подвергается переделкам и не лишается имеющихся возможностей. Такой терминал не будет уникальным и дорогим, поскольку речь не идет о создании и организации производства нового устройства, а лишь о модификации существующего. Успешный опыт реализации автономного ("карманного") терминала на базе органайзера PSION XP-II [3] убедительно подтверждает это.

Отметим, что "Наставник" на автономных терминалах приобретает дополнительные полезные качества: возможность работы вне класса, удаленная загрузка курсов через Интернет и др. Кроме того, использование модифицированных устройств "автоматически" способствует популяризации системы. До сих пор желание иметь свой "Наставник" возникало, как правило, лишь после работы в нем. Тех, кому удалось это сделать, было немного. Теперь же, купивший модифицированный калькулятор (записную книжку, органайзер, телефон...), может заинтересовать его новыми качествами и пожелать их использовать. Для укрепления этого желания целесообразно иметь

"зашитым" в устройство хорошо составленный курс, обучающий работе с ним самим.

Описываемый далее автономный терминал "Наставника", реализованный на калькуляторе "Электроника МК-85", представляет собой вторую успешную попытку развития системы в новом, перспективном направлении. Помимо того, что в основе терминала лежит отечественное изделие, реализация интересна тем, что выполнена существенно отличным от своего предшественника способом.

Специфика реализации терминала на "ширпотребе"

Устройства, пригодные для модификации под "Наставник", живут недолго, обычно не более 2-3 лет, а их следующие модели как правило настолько несовместимы с предшественниками, что в них без изменений нельзя применять программы для текущей модели. Их разработчики не ставят перед собой такой цели, в отличие от разработчиков универсальных компьютеров.

Серийный производитель, как правило, работает независимо от разработчика и не имеет с ним прямого контакта. Документация, поставляемая с устройством, и та, что имеется у производителя, не содержит и не должна содержать сведений, необходимых для внесение изменений в устройство. Даже если удается наладить контакт с разработчиками устройства, это бывает мало продуктивным – они уже заняты другими делами и не хранят документацию по прошлым разработкам, т.к. сопровождать ширпотреб не имеет смысла – проще и дешевле выпустить на рынок новую модель.

Еще одна проблема – ограниченность ресурсов устройства, прежде всего памяти для программ и данных. Небольшой резерв памяти обычно имеется и нужно в него уместиться. Хотя в последние годы острота этой проблемы заметно снизилась, для МК-85 вопрос с памятью все еще актуален.

Возможны два подхода к реализации «Наставника» на серийно выпускаемом устройстве. Первый был опробован на органайзере PSION XP-II [3], программирование на котором было возможно на встроенном процедурном языке OPL. Выбор таких "открытых" устройств весьма ограничен. Другой подход – расширение функциональных возможностей устройства путем его доделки, зачастую вмешательства в «штатную» работу. Устройство при этом должно сохранить свою функциональность. Именно этот подход оказался приемлем для МК-85.

Суть его состоит в следующем. За основу был взят однотерминальный вариант «Наставника», написанный на языке диалоговой системы структурированного программирования ДССП [4].

ДССП представляет собой интерпретирующую систему со своим машинно-независимым языком программирования. Система построена методом «раскрутки»: небольшое машинно-зависимое ядро, содержащее набор базовых команд-примитивов и интерпретатор внутреннего представления программ – сшитого кода, реализует язык ДССП нижнего уровня. Этот язык, набор слов-операций, можно назвать языком ассемблера ДССП-машины. На этой основе определены слова более высокого уровня, и т.д. Вся остальная часть системы (средства разработки программ и средства поддержки диалога – интерпретатор входного языка ДССП, компилятор программ во внутреннее представление, отладчик, редактор текстов) написана на своем собственном языке.

В качестве первого шага из всей совокупности подсистем МСО «Наставник» была взята наиболее важная часть – подсистема обучения и использовано имеющееся ядро ДССП для микропроцессора с архитектурой PDP-11 (на базе такого процессора сделан МК-85). Метод построения готовой системы ранее был неоднократно опробован при переносе самой ДССП на различные компьютеры [8].

На полномасштабном компьютере, где достаточно памяти, ДССП-программа обычно исполняется в среде системы разработки. В целевых системах чаще ограничиваются удалением крупных компонент – редактора текстов и отладчика. В нашем случае этого было явно недостаточно, поэтому было решено воспользоваться «целевой компиляцией», т.е. оставить в готовой системе только целевую программу и ядро ДССП, исключив оттуда интерпретатор входного потока, словарь, компилятор и прочие компоненты.

Особенности реализации терминала на калькуляторе МК-85

МК-85 представляет собой типичную микрокомпьтерную структуру, содержащую микропроцессор с системой команд PDP-11, ПЗУ 32КВ, статическое ОЗУ емкостью до 6КВ и контроллер матричного ЖК-дисплея. В микропроцессор встроен контроллер 54-клавишной клавиатуры и 15-разрядный параллельный I/O порт, подключиться к которому рядовому пользователю нельзя. В младшей половине ПЗУ (16КВ) размещена программа, реализующая БЭЙСИК, со встроенными в нее драйверами клавиатуры и дисплея. Старшая половина ПЗУ свободна.

Наличие в МК-85 свободной программной памяти дало возможность разместить дополнительную компоненту ПО в ней, а наличие БЕЙСИК-а - использовать имеющиеся драйверы клавиатуры и дисплея. Основное препятствие на этом пути состояло в том, что

отсутствовала достоверная документация на ПО МК-85. Другое препятствие - отсутствие у МК-85 "связи с внешним миром". Для автономного терминала такая связь (в виде, например, COM-порта) желательна. В описываемой реализации достоверные коды БЕЙСИК-а получены чтением через программатор содержимого штатного ПЗУ, а COM-порт предполагается программно эмулировать на имеющемся параллельном порте МК-85 путем его доработки.

После дизассемблирования всего кода программы и его изучения удалось найти процедуры чтения кода нажатой клавиши и вывода литеры в определенное знакоместо на индикаторе МК-85. Эти две процедуры обеспечивают интерфейс обучаемого с системой.

Основные этапы сборки и отладки системы таковы: написанное на языке ассемблера машинно-зависимое ядро ДССП компилируется, получается код ядра. Для этого используется кросс-ассемблер для микропроцессора PDP-11, созданный ранее в ДССП [7].

Программа-компоновщик собирает подсистему обучения МСО «Наставник» из кода ядра ДССП и текста программы «Наставника», переводя его во внутреннее представление. Компоновщик также был создан специально для этих целей на основе имевшихся разработок штатного компоновщика ДССП и компоновщика, применявшегося для сборки ДССП, написанной на языке Си [6].

Простая программа-построитель образа ПЗУ формирует код прошивки для записи в ПЗУ калькулятора:

- укладывает в этот образ исходный код МК-85, код «Наставника», управляющую информацию по обучающим курсам;
- в свободную позицию таблицы переключения режимов на МК-85 прописывает ссылку на начало «Наставника»;
- сохраняет полученный образ.
- Для целей отладки из калькулятора была выведена колодка, в которую вставляется сменное ПЗУ. Таким образом можно было легко опробовать различные варианты. Способ далеко не самый оперативный, но необходимый по крайней мере на этапе отладки ввода-вывода.

Предварительная отладка «Наставника» проводилась в ДССП с применением простого эмулятора «окружения», предоставляемого калькулятором. Окончательная отладка алгоритма проводилась как на самом МК-85, так и на программном эмуляторе PDP-11 [5].

Оснащение МК-85 асинхронным COM-портом

Наличие связи с внешним миром для автономного терминала "Наставника" желательно. Конечно, можно изначально “зашить” в него некоторый набор курсов и этим ограничиться. Однако такая система лишится одного из основных достоинств - наличия обратной связи с

автором курса. Многие готовые устройства-прототипы (такие, например, как телефонные аппараты) имеют встроенные связные возможности. Их и следует выбирать в качестве основы автономного терминала. В нашем случае (ориентация на отечественное изделие) выбирать было не из чего - МК-85 не имел ни штатного COM-порта, ни даже доступа к выводам имеющегося параллельного порта. Поэтому была проведена доработка параллельного порта путем преобразования TTL сигналов последнего в сигналы стандарта RS-232C, используемых в COM-портах PC. Программная эмуляция UARTa (асинхронного приемника/передатчика) обычно основывается на том, что стандартная длительность старт-стопных посылок формируются путем задания числа циклов в программе. В данном случае этот способ оказался непригодным, поскольку тактовая частота процессора в МК-85 не стабилизирована кварцевым резонатором. Поэтому длительность старт-стопных посылок имеет значительный разброс у разных экземпляров и нестабильна во времени даже у одного и того же. Для преодоления этого недостатка программа, эмулирующая UART, должна быть адаптивной - уметь автоматически подстраиваться под конкретное значения скорости процессора. Для этого в начале сеанса связи МК тестирует входную линию и ждет изменения ее состояния. PC посылает байт, состоящий из одних нулей. Настраивающая программа ждет возврата линии в 1 и считает циклы. Получившееся число делиться на 9. Т.о. определяется число циклов, соответствующее текущей длительность бита. Затем PC посылает байт, состоящий из чередующихся 1 и 0, а программа приема тестирует линию с интервалом в один бит. Приняв байт, его посылают назад в PC, формируя старт-стопную посылку программно, т.е. засылая 0 и 1 с требуемым интервалом в выделенный разряд параллельного порта. Если обнаруживается несовпадение, процесс повторяется с меньшей скоростью. Установление связи и настройка скорости обмена должны происходить в каждом сеансе связи, т.к. временные параметры могут меняться, например, с нагревом аппаратуры.

Объем программы, обеспечивающей такой обмен, менее 1 КБ.

Выводы

Получившийся вариант подсистемы обучения МСО «Наставник» оказывается вполне жизнеспособным, несмотря на весьма ограниченные ресурсы микрокалькулятора. После разработки программы обмена по последовательному порту подсистема обучения займет около 5 КБ, для управляющей информации по обучающим курсам останется 11 КБ, что достаточно для 3-5 курсов. Из имеющихся

6 КБ ОЗУ занято менее 0.5 КБ, остальную память можно использовать для загрузки управляющей информации из компьютера или для хранения протокола занятия.

Время реакции системы составляет доли секунды.

В то же время дальнейшее продвижение этой системы связано с проблемами, лежащими вне научных исследований. Поскольку необходима доработка серийного устройства, требуется участие разработчика и производителя.

Сегодня появился "ширпотреб" особого рода: мобильные телефоны и карманные PC, имеющие встроенный интерпретатор Java, сменные карты flash-памяти на десятки мегабайт, штатные средства связи с компьютером, в том числе беспроводные – инфракрасный порт. Стоимость их превышает стоимость калькуляторов, но через год-полтора они окажутся доступными массовому потребителю. Новая реализация МСО «Наставник» должна ориентироваться на них.

Литература

1. Брусенцов Н.П., Маслов С.П., Рамиль Альварес Х. Микрокомпьютерная система обучения "Наставник" - М.: Наука, 1990. - 223с.

2. Брусенцов Н.П., Маслов С.П., Рамиль Альварес Х. Методическое пособие по разработке учебных материалов в микрокомпьютерной системе обучения "Наставник" - М.: Изд-во Моск.ун-та, 1992. - 95с.

3. Маслов С.П. Карманная АСО "Наставник" (стартовая версия). // Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001. С.173-183.

4. Брусенцов Н.П., Захаров В.Б., Руднев И.А., Сидоров С.А., Чанышев Н.А. Развиваемый адаптивный язык РАЯ диалоговой системы программирования ДССП. М, Изд-во МГУ, 1987. –80 с.

5. ERSATZ-11 PDP-11 Emulator, 2000. www.dbit.com

6. Бурцев А.А., Франтов Д.В., Шумаков М.Н. Разработка интерпретатора сшитого кода на языке Си. //Вопросы кибернетики. Средства разработки внутреннего программного обеспечения. Микроэлектроника. Перепрограммируемые системы. М., 1999. С.64-76.

7. Sidorov S.A. Assemblers for firmware systems. // EuroFORTH'99: 15th Euroforth Conference, St.Pitersburg, Russia, 1999. pp.18-26.

8. Сидоров С.А. Исследование переносимости ДССП. // Диалоговые микрокомпьютерные системы, М, Изд-во МГУ, 1986. С. 30-37.

Акишин Р.С.

Построение индикатора рынка на основе нейросетевого подхода. Использование однородной Марковской модели для анализа выходов нейросети в задачах классификации

Цель этой работы – рассмотрение вопросов применения однородной Марковской модели для анализа выходов нейросети в задачах классификации на основе временных рядов. Такие задачи, обычно, возникают при построении экспертных систем в финансовой области. Например, основой успешной работы на бирже, является правильное понимание того, в какой стадии находиться рынок. Цены могут повышаться, понижаться либо находиться в рамках торгового диапазона (т.е. колебаться около одного значения). Исходя из этого, для оценки рынка надо применять те или иные методы технического анализа. Например, правило пересечения скользящих хорошо работает при выраженной тенденции, но не пригодно в ситуации, когда цены колеблются в рамках торгового диапазона (рис.1). «На глаз» определить текущую тенденцию на рынке в большинстве случаев, даже для опытного аналитика, является нетривиальной задачей. В статье исследования проводятся на примере применения однородной Марковской модели для анализа выходов нейросети в задаче построения индикатора рынка.

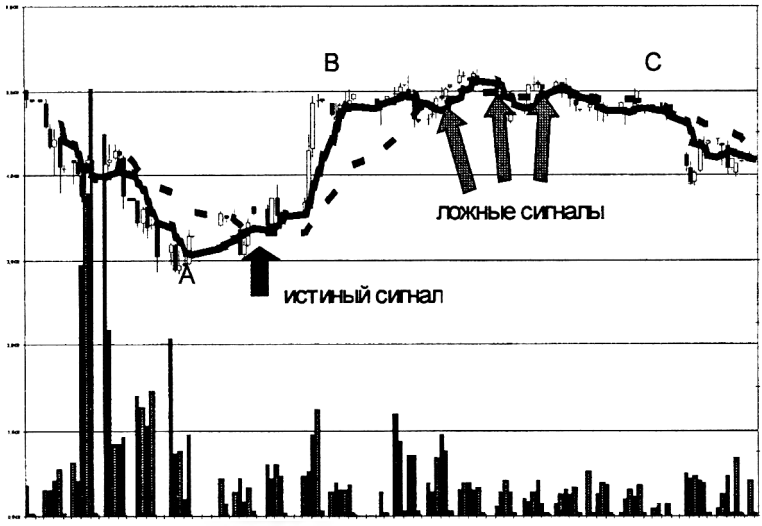


Рисунок 1. ▬ – скользящая mA(5), ■ ■ ■ – mA(17). Акции компании «IBM». Отрезок BC – колебания цен в рамках диапазона.

В техническом анализе для решения этой задачи существует несколько методик. Наиболее популярные из них основаны на выявлении последовательностей ценовых экстремумов, анализе скользящих средних, изучении графиков «японские свечи» (более подробно см. [1]). Но всем им присуще следующие недостатки:

1. *Запаздывание.* Любая из методик фактически определяет, какая тенденция началась несколько периодов назад и, возможно, продолжается сейчас.
2. *Нет возможности выявить ситуацию торгового диапазона.*
3. Как следствие из предыдущих пунктов, в ситуации торгового диапазона *могут появляться ложные сигналы* изменения тенденции.

Предложим к рассмотрению «идеальный» индикатор, генерирующий *без запаздывания* три сигнала: цены повышаются (назовем K_1), цены понижаются (K_{-1}), цены колеблются в рамках торгового диапазона (K_0). Фактически, задача построения такого индикатора сводиться к задаче классификации на основе значений цен $x(t)$, которая (если подходить формально) решается в четыре этапа [2]:

Формируется алфавит признаков для описания распознаваемых объектов в виде строк признаковых описаний.

Формируются описания каждого класса объектов в виде выборок описаний объектов - прецедентов.

По описаниям классов создается распознающий алгоритм.

Алгоритм решает задачу распознавания для произвольного нового объекта по его признаковому описанию.

Для описания распознаваемых объектов можно использовать различные параметры. В рассматриваемой задаче можно взять следующие величины: $x(t).close$ (цена закрытия для периода t), $x(t).open$ (цена открытия), $x(t).volume$ (объем сделок) – за последние m периодов (окно выборки). Причем, для каждого окна все значения необходимо нормировать в сегмент [0,1].

Рассмотрим следующее описание классов:

Будем говорить, что объект $x(t) \in K_1$, если $\frac{x(t+n).close}{x(t).close} > 1 + \sigma \sum_{i=0}^{n} \frac{1}{2^n}$, где σ и n – настраиваемые параметры (n – количество рассматриваемых периодов для классификации, σ – процент возможного колебания цены в рамках диапазона за один период). Так как максимальная амплитуда колебаний цен в диапазоне достигается за несколько периодов и впоследствии может лишь незначительно увеличиваться, здесь используется сумма $1 + \frac{1}{2} + \frac{1}{4} + ... + \frac{1}{2^n} = \sum_{i=0}^{n} \frac{1}{2^n} = 2 - \frac{1}{2^n}$.

Аналогично, объект $x(t) \in K_{-1}$, если $\frac{x(t+n).close}{x(t).close} < 1 - \sigma \sum_{i=0}^{n} \frac{1}{2^n}$.

Если ни одно из неравенств не выполнено, то объект $x(t) \in K_0$.

Построим распознающий алгоритм на базе нейросетевого подхода. Обычно, для задач классификации используют сети прямого распространения. Для выбора архитектуры нейросети (количества нейронов и связей между ними) можно воспользоваться, например, *конструктивным* алгоритмом построения сети из заданного класса. Класса однослойных сетей для решения данной задачи в общем случае не достаточно: так для котировок компании «Лукойл» с параметрами $m = 10$, $n = 6$ среднеквадратичная ошибка обучения (MSE) равна 0.34. Так как показано [3], что произвольное отображение можно реализовать с помощью двухслойной нейросети, то эффективную архитектуру и следует искать в этом классе.

Описание классов будем кодировать следующим образом:

	K_1	K_0	K_{-1}
OUT_1	$0.8 \pm \xi_n$	$-0.8 \pm \xi_n$	$\pm 2\xi_n$
OUT_2	$0.8 \pm \xi_n$	$\pm 2\xi_n$	$-0.8 \pm \xi_n$
OUT_3	$\pm 2\xi_n$	$0.8 \pm \xi_n$	$-0.8 \pm \xi_n$

Где ξ_n – последовательность случайных нормально распределенных величин: $E(\xi_n) = 0$, $D(\xi_0) = 0.2$, $D(\xi_n) \xrightarrow{n} 0$.

Такое кодирование (2-на-2), по сравнению с классическим, позволяет реализовывать более сложные разбиения классов [4]. Добавление случайной компоненты позволяет увеличить скорость обучения нейросети. А смещение целевых значений от 1 и -1 к нулю позволяют более эффективно тренировать нейросеть, путем введения «штрафа» на неограниченное увеличение весов на последнем слое.

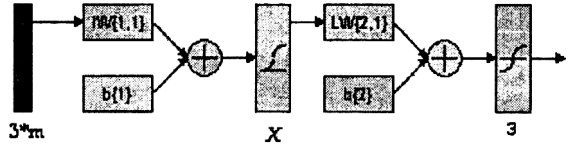


Рисунок 2. Двухслойный

Исходя из описаний распознаваемых объектов и классов, в качестве функций активации следует использовать: $logsig(n) = \frac{1}{1+e^{-n}}$ для первого слоя и $tansig(n) = \frac{2}{1+e^{-2n}} - 1$ – для второго (рис. 3).

При применении вышеописанного нейросетевого алгоритма сталкиваемся с проблемой: каким образом оценивать выход сети $out = (out_1, out_2, out_3)$? Можно построить проекции точки out на плоскости $OUT_1 \times OUT_2$, $OUT_1 \times OUT_3$, $OUT_2 \times OUT_3$, на них

оценить расстояния до каждой цели $\rho_{K_1} = \rho(\Pr out, K_1)$, $\rho_{K_0} = \rho(\Pr out, K_0)$, $\rho_{K_{-1}} = \rho(\Pr out, K_{-1})$ и выбрать минимальное. Но часто складывается ситуация, что расстояния хотя бы до двух целей практически равнозначны (т.е. $\exists i, j : i \neq j, \left|\rho_{K_i} - \rho_{K_j}\right| < \varepsilon$). Тогда необходимо для классификации использовать дополнительную информацию. В качестве таковой, рассмотрим матрицу переходов однородной Марковской модели $M = (m_{ij})_{3\times3}$, $m_{ij} = P(x(t) \in K_i \mid x(t-1) \in K_j)$, которую можно построить на основании обучающей выборки.

Введем преобразованные расстояния до целей. Пусть $x(t-1) \in K_i$, тогда для $x(t)$ получим $\hat{\rho}_{K_j} = \rho_{K_j} * (1 - m_{ij})$. Выбрав из $\hat{\rho}_{K_j}$ минимальное значение, определяем принадлежность объекта $x(t)$ к одному из классов.

Рисунок 3. Появление "неопределенных точек" при использовании Марковской модели

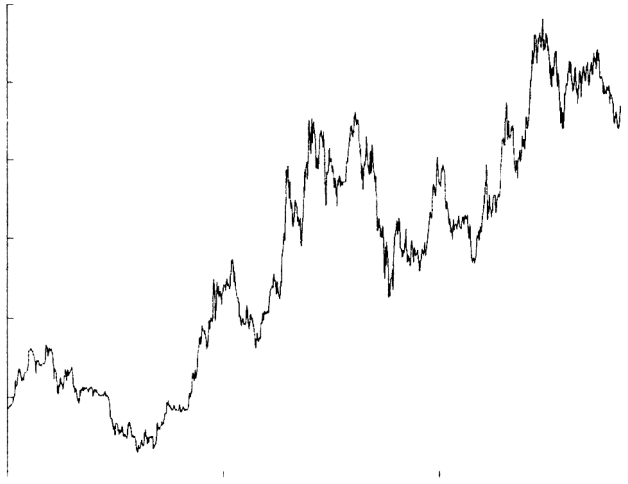


Рисунок 4. Акции компании "Лукойл". ММВБ. 05.97-02.03.

При использовании однородной Марковской модели возникает другая проблема: для классификации $x(t)$ необходимо знать принадлежность объекта $x(t-1)$. На практике, нам не известна принадлежность $(n-1)$ последних объектов (так называемых «неопределенных точек» (рис.3)). Для их классификации построим нейросети такой же архитектуры. Единственное отличие будет заключаться в окне входных данных. Оно будет несколько смещено вперед.

Рассмотрим результаты работы данной модели для котировок акций компании «Лукойл» за последние шесть лет (более 1400 периодов; источники [5, 6]). Как видно из рис. 4, на данном рынке присутствуют все классы примерно в одинаковой пропорции ($|K_1| \approx 0.37$, $|K_0| \approx 0.29$, $|K_{-1}| \approx 0.34$).

Для тестирования выделим из выборки случайным образом 40% точек, и на оставшихся данных обучим нейросеть.

При $m = 10, n = 6$ получим следующие ошибки классификации[1]:

---

[1] Результаты получены в системе Matlab 6.1. Нейросети обучались с помощью алгоритма SCG Моллера на протяжении 10000 эпох. Обучение одной такой нейросети на процессоре Pentium II 450 занимает по времени около 40 минут.

Ошибки классификации неопределенных объектов					Ошибка классификации цели $x(t)$
$x(t-5)$	$x(t-4)$	$x(t-3)$	$x(t-2)$	$x(t-1)$	
15%	31.58%	44.08%	53.29%	55.26%	55.76%

С другой стороны, если мы *точно* знаем принадлежность «неопределенных точек», ошибки классификации составляют:

Ошибки классификации неопределенных объектов					Ошибка классификации цели $x(t)$
$x(t-5)$	$x(t-4)$	$x(t-3)$	$x(t-2)$	$x(t-1)$	
15%	16.11%	18.2%	17.19%	16.11%	17.32%

То есть ошибка классификации в момент $(t-1)$ влечет за собой ошибку в момент t (проявляется эффект «домино»).

Для уменьшения ошибки классификации «неопределенных точек» (и, в итоге, точки $x(t)$) можно использовать несколько[1] нейросетей следующим образом:

[1] Для точки $x(t)$ можно применить $(t_0 - t + 1)$ ранее обученных нейросетей.

Пусть $\forall t \in [t_0 - n + 1 \;\; t_0]$, $N = t_0 - t + 1$, $\forall i = \overline{1,N}$, $res = \begin{cases} res + 1, \text{ если } x_i(t) \in K_1 \\ res - 1, \text{ если } x_i(t) \in K_{-1}, \\ res, \text{ иначе} \end{cases}$

$ares = \begin{cases} ares + 1, \text{ если } x(t), \notin K_0 \\ ares, \text{ иначе} \end{cases}$, $\Psi_1 = ares \le \left\lfloor \frac{N}{2} \right\rfloor$, $\Psi_2 = res > \left\lfloor \frac{N}{2} \right\rfloor$, $\Psi_3 = res < -\left\lfloor \frac{N}{2} \right\rfloor$

тогда

$\Psi_1 \vee \overline{\Psi_2} \wedge \overline{\Psi_3} \Rightarrow x(t) \in K_0$

$\overline{\Psi_1} \wedge \Psi_2 \Rightarrow x(t) \in K_1$

$\overline{\Psi_1} \wedge \Psi_3 \Rightarrow x(t) \in K_{-1}$.

В этом случае ошибка классификации составит:

Ошибки классификации неопределенных объектов					Ошибка классификации цели $x(t)$
$x(t-5)$	$x(t-4)$	$x(t-3)$	$x(t-2)$	$x(t-1)$	
14.8%	21.93%	29.82%	34.21%	34.65%	35.77%

Таким образом, получаем приемлемую точность для данной задачи. Для сравнения: ошибка классификации для этой задачи при использовании метода ***K***-соседей ($K = 10$) составляет 62%; ***K***-эталонов – 60%; Марковских цепей – 54%.

В заключение необходимо отметить, что описанный выше алгоритм не является единственным способом анализа выходов нейросети. Для этого можно использовать различные параметры, которые характерны для каждой конкретной модели, что в большинстве случаев будет давать не менее точный результат. Например, в рассмотренной задаче можно использовать такие инструменты технического анализа, как осцилляторы. Но их применение осложнено проблемами выбора таких параметров и синтеза решающих правил.

Литература

1. Дж. Швагер «Технический анализ. Полный курс». – М.: Альпина Паблишер, 2001.
2. Рязанов В.В. «Нестатистические методы классификации. Курс лекций».
3. S. Haykin, «Neural Networks. A Comprehensive Foundation». – NY.: Macmillan College Publishing Company, Inc., 1994.

4. Д.-Э. Бэстенс «Нейронные сети и финансовые рынки: принятие решений в торговых операциях». – М.: ТВП, 1997.
5. ИК Ленмонтажстрой, Котировки акций 05.97 - 08.99 http://www.lmsic.com/quotes/market.php.
6. ИК Финанс-Аналитик, Котировки акций 08.99 - 02.03 http://www.finam.ru/analysis/export/default.asp.

Карганов К.А.
Организация интерфейса отладчика для параллельных программ.

Введение

Одной из самых трудоемких стадий разработки параллельных программ является отладка. Опрос, проведенный среди разработчиков параллельных программ на конференции Supercomputing'93, показал, что написание и отладка кода программы, в среднем занимают около половины всего времени разработки приложения [1]. При этом около трети разработчиков не пользуются какими-либо программными инструментами для отладки и оптимизации параллельных программ.

Более поздние работы [2,3] отмечают тот факт, что несмотря на огромный прогресс в развитии элементной базы и программного обеспечения современных вычислительных систем, средства для отладки приложений остались примерно на том же уровне, что и 30 лет назад. Отладка сложных программных систем все еще остается скорее искусством, нежели частью стандартного цикла промышленной разработки.

Для параллельных приложений зачастую характерны как значительная сложность алгоритма и большой объем данных, характеризующий состояние программы, так и недетерминизм выполнения программы и множество специфичных труднообнаруживаемых ошибок. Эти и многие другие факторы [3,4,5] приводят к тому, что на протяжении вот уже более 15 лет проблема отладки параллельных приложений сохраняет свою актуальность.

При разработке параллельного отладчика возникает большое количество проблем. Одной из них является проблема грамотной организации пользовательского интерфейса процесса параллельной отладки. Организация пользовательского интерфейса значительным образом влияет на удобство и эффективность работы с отладчиком. В отличие от отладки последовательных программ, до сих пор не существует достаточно универсальных и общепринятых подходов к организации интерфейса отладчика для параллельных программ, особенно для систем с распределенной памятью.

Данная статья предлагает один из подходов к решению указанной проблемы. Этот подход реализован в отладчике mpC Workshop [6] для среды программирования mpC [7], разработанных в Институте Системного Программирования Российской Академии Наук. Статья организована следующим образом: раздел 1 дает обзор основных

типов отладчиков, в разделе 2 описываются основные проблемы построения параллельных отладчиков, главным образом, с точки зрения организации пользовательского интерфейса. В разделе 3 представлен краткий обзор основных подходов к организации пользовательского интерфейса на примере наиболее известных отладчиков для параллельных программ. Раздел 4 дает краткий обзор основных идей языка mpC, а раздел 5 описывает подход к организации интерфейса, использованный в отладчике mpC Workshop и сравнивает его с другими известными подходами.

1. Основные типы отладчиков

Согласно устоявшейся терминологии, отладчик – это инструмент, позволяющий обнаруживать, локализовывать и исправлять ошибки в программах. В настоящее время различают несколько основных методов отладки (реальные отладчики зачастую используют комбинацию этих методов): статический анализ, анализ событий, интерактивная отладка, Post-mortem отладка. Развернутый обзор с примерами конкретных отладчиков дан в работе [5], здесь же приведен краткий обзор вышеназванных методов.

Статический анализ представляет собой исследование исходного текста программы и включает в себя анализ потока управления и потока данных и анализ графа вызовов процедур и функций. Метод хорошо применим к отладке параллельных программ, но имеет множество ограничений, связанных со сложностью моделирования динамики выполнения программы.

Анализ событий (event-based debugging) представляет собой описание некоторого набора событий в программе (набор некоторых условий) и реакций на эти события (например, отображение данных или изменение состояния программы). Часто используемая разновидность такого метода – сохранение трассы выполнения программы и ее анализ после выполнения [3], сравнение с другими трассами или даже «проигрывание» сессии выполнения программы. Частным случаем метода является «посмертная» (post-mortem) отладка путем изучения трассы и журнала выполнения программы после того, как программа полностью отработала. Эти методы широко используются, в первую очередь, для отладки параллельных программ, поскольку слабо влияют на динамику выполнения приложения и дают возможность детального изучения конкретного прогона программы, позволяя легко обнаруживать многие труднообнаруживаемые ошибки (например, ошибки синхронизации или обмена сообщениями).

Интерактивная отладка – наиболее распространенный и привычный способ отладки, особенно для отладки последовательных

программ, представляет собой циклический процесс запуска и выполнения программы с целью «ручного» обнаружения ошибок путем останова выполнения программы в определенных местах, изучения состояния программы и пошаговой трассировки «подозрительных» участков кода. Этот метод очень хорошо подходит для отладки последовательных программ, т.к. он понятен и удобен для использования, однако при попытке применить его к отладке параллельных программ возникает множество проблем, связанных как со сложностью параллельной программы, так и с невозможностью повторения трассы программы в общем случае (недетерминизм выполнения). Все это приводит к тому, что создание интерактивных параллельных отладчиков представляет собой сложную, но интересную задачу.

2. Проблемы организации интерфейса интерактивных отладчиков для параллельных программ

Рассматривая процесс интерактивной отладки, можно выделить два основных типа отладочных действий: управление выполнением программы и исследование внутреннего состояния отлаживаемой программы. Даже для последовательной программы полное состояние включает в себя состояние сегментов данных, кода и стека, состояние регистров, открытых файлов, среды выполнения и т.д. – огромный объем информации, который невозможно охватить и понять целиком.

Поэтому основной задачей пользовательского интерфейса отладчика является ограничение объема отображаемой информации, представление информации в удобной и компактной форме, предоставление необходимых средств для управления отображением информации а также для управления собственно выполнением отлаживаемой программы. Как правило, современные отладчики используют графический пользовательский интерфейс (GUI), поскольку он позволяет отображать больше информации, организовывать отображение более удобным способом а также предоставляет простые средства для настройки и управления отображением.

Для отладки последовательных программ принципы организации интерфейса отладчиков общеизвестны. В качестве примера можно привести продукты фирмы Borland или Microsoft Visual Studio. Для параллельных отладчиков это не так, в первую очередь, из-за растущего противоречия между желаемой простотой и удобством управления программой в отладчике и внутренней сложностью, присущей параллельной программе.

Если параллельная программа состоит из N процессов, то объем отображаемой информации, характеризующей состояние

программы, превосходит объем информации в последовательном случаее более чем в N раз – появляется информация, специфичная для параллельной программы, такая как очереди сообщений, времена и параметры коммуникаций, информация о распределении данных и вычислений и т.д. Предполагая, что N может быть достаточно большим, получаем требование масштабируемости интерфейса отладчика.

Для того, чтобы по возможности упростить использование отладчика, должны быть предусмотрены удобные средства управления выполнением программы и отображением данных. Эти средства должны быть достаточно гибкими, чтобы ограничить отображение избыточной информации и в то же время позволять быстро фокусироваться на той информации, которая интересует пользователя в конкретный момент времени.

Взаимная противоречивость требований к параллельному отладчику приводит к тому, что организация пользовательского интерфейса отладчика представляет собой очень непростую задачу, которую вынуждены решать все разработчики подобных отладчиков. Множество существующих на данный момент отладчиков для параллельных приложений предлагают множество различных способов решения этой проблемы, но, тем не менее, до сих пор нет единого, общепринятого подхода к организации интерфейса параллельной отладки.

3. Обзор существующих подходов

В качестве примера рассмотрим три мощных интерактивных отладчика – TotalView [8], Prism[9,10] и p2d2[11], обращая основное внимание на вопросы организации пользовательского интерфейса и соответствия его вышеперечисленным требованиям.

TotalView. Один из самых мощных и универсальных параллельных отладчиков – TotalView [8] от компании Etnus. Он разработан как универсальный инструмент, позволяющий отлаживать как последовательные, так и параллельные программы и поддерживает множество языков и библиотек параллельного программирования.

Поскольку отладчик предназначен для отладки любых программ, он не делает никаких предположений о структуре исходного текста программы и поэтому открывает текст каждого процесса в отдельном окне отладчика. Это самый универсальный, но, пожалуй, самый неудобный для пользователя подход. Он позволяет отобразить максимальное количество информации о состоянии каждого процесса, но усложняет управление отладкой.

Отладчик имеет корневое окно, отображающее список всех процессов параллельной программы с краткой информацией об их

состоянии, окно управления группами процессов и позволяет открывать окно с детальной информацией (текст, стек, данные) для каждого конкретного процесса. Также имеется большой набор инструментов для визуализации дерева вызовов (расширение понятия стека вызовов для параллельного случая), очередей сообщений и различных режимов визуализации данных.

Отладчик предоставляет очень мощные возможности для изучения состояния программы и поддерживает множество платформ и языков, но из-за своей универсальности имеет низкую масштабируемость, не позволяет быстро переключаться на отображение нужных данных и предоставляет довольно сложный и неудобный интерфейс.

Prism. Другой известный отладчик – Prism [9,10], входящий в пакет Sun Cluster Tools от Sun Misrosystems. Этот отладчик изначально был спроектирован для достижения максимальной масштабируемости и обеспечения интерфейса, близкого к последовательному отладчику. Он имеет мощный язык управления отладкой из командной строки, графический пользовательский интерфейс и множество разнообразных инструментов для визуализации данных.

Prism вводит новое понятие: *pset* – группа процессов, которая может использоваться как единое целое. Пользователь может определять произвольные группы и работать с группами, вместо отдельных процессов. Большинство команд отладчика (в том числе и отображение данных) могут применяться к таким группам. Pset может зависеть от состояния программы и изменяться динамически во время отладки. Это дает очень мощный и гибкий способ управления большими группами процессов.

Отладчик имеет средства для отображения дерева вызова функций, динамических структур данных, очередей сообщений, множество режимов отображения массивов, развитое управление событиями, однако, в силу исторических причин, большинство функций отладчика доступно только из командной строки. Развитый командный язык с возможностью задания собственных команд открывает широкие возможности управления процессом отладки, но усложняет освоение и использование отладчика.

P2d2. Оригинальный подход реализован в отладчике p2d2 [11] (сокращение от Portable Parallel/Distributed Debugger) от NASA. Отладчик предоставляет три уровня детализации представления информации – все процессы программы, текущая группа процессов (focus group) и текущий процесс (focus process). Все три уровня отображаются одновременно, что позволяет с одного взгляда оценить состояние программы, не перегружает интерфейс избыточными данными и позволяет легко переключаться на нужную информацию.

На самом верхнем уровне абстракции отображается матрица процессов. Каждый процесс отображается в виде иконки, отображающей информацию о состоянии процесса, определенную пользователем. Такой подход позволяет видеть в общих чертах состояние всех процессов программы, и быстро выбирать процессы для более детального изучения.

На следующем, более детальном, уровне абстракции текущая группа отображает состояния процессов более детально. С каждым процессом связана текстовая строка, отображающая в заданном формате более детальную информацию о процессе, такую как идентификатор процесса, состояние выполнения и текущую позицию в исходном тексте программы.

Самая подробная информация выводится о текущем процессе. Главное окно отладчика отображает исходный текст процесса с отмеченной текущей позицией и стек вызовов функций для данного процесса. Для текущего процесса также отображаются значения его переменных. Для облегчения сравнения значений переменных на разных процессах вводится понятие дополнительного текущего процесса – в окне вывода данных выводятся значения переменных для двух указанных процессов.

Такой подход обеспечивает среднюю масштабируемость, но быстрое переключение на необходимые данные, а также простой и понятный пользовательский интерфейс.

4. Основные идеи языка mpC

Основная идея системы программирования mpC [7] заключается в том, что mpC-приложение явно описывает топологию абстрактной сети, по которой распределяются данные, коммуникации и вычисления. Система программирования использует эту информацию для отображения этой топологии на реальную сеть, на которой выполняется параллельная программа, таким образом, чтобы обеспечить эффективное выполнение приложения на этой сети. Это отображение производится во время выполнения программы, основываясь на информации о производительности узлов и каналов связи реальной вычислительной системы, динамически адаптируя программу к конкретной вычислительной системе.

Язык mpC представляет собой расширение языка ANSI C, которое вводит новый тип ресурса – вычислительное пространство, определяемое как набор виртуальных процессоров различной производительности, соединенных каналами различной пропускной способности. Во время выполнения реальные процессы исполняемой параллельной программы представляют собой виртуальные

процессоры. Программист управляет вычислительным пространством с помощью создания и удаления областей вычислительного пространства, называемых сетевыми объектами или сетями, аналогично управлению объектами данных (областями памяти). Сети в mpC используются для распределения данных, вычисления выражений и выполнения операторов.

Таким образом, mpC представляет собой высокоуровневый язык параллельного программирования, позволяющий писать эффективные программы для наиболее общего класса параллельных систем – неоднородных вычислительных систем. Язык mpC упрощает разработку параллельных приложений, сравнимых по эффективности с высококачественными MPI [12] приложениями, а на неоднородных системах показывает более высокие результаты, чем MPI аналоги [13]. Более подробная информация о системе программирования mpC на сайте http://www.ispras.ru/~mpc.

5. Отладчик mpC Workshop

При проектировании отладчика для языка mpC были поставлены следующие требования: отладчик должен быть интерактивным и иметь интерфейс, как можно ближе к традиционному последовательному стилю отладки, отладчик рассчитан только на отладку программ на языке mpC, причем только в терминах исходного текста программы, отладчик должен иметь хорошо масштабируемый интерфейс, позволяющий отлаживать большое количество параллельных процессов, и быть удобным в использовании.

Поскольку семантика пересылок в языке mpC синхронная и в нем нет недетерминированных коммуникаций, таких как «принять сообщение от любого процесса», то параллельные программы на mpC являются детерминированными в отношении порядка коммуникаций между процессорами. Это означает, что любая трасса выполнения программы повторяема с точки зрения логического времени коммуникаций [14] в программе. Это свойство языка mpC делает возможным интерактивную отладку программ без необходимости сохранения и воспроизведения трассы коммуникаций.

Основное окно интегрированной среды разработки mpC Workshop показано на рис. 1. Главное окно отображает исходный текст программы, управляющие объекты, такие как точки останова и текущие позиции процессов параллельной программы в исходном тексте. Дополнительные окна отображают файлы проекта, значения переменных программы и множество другой информации.

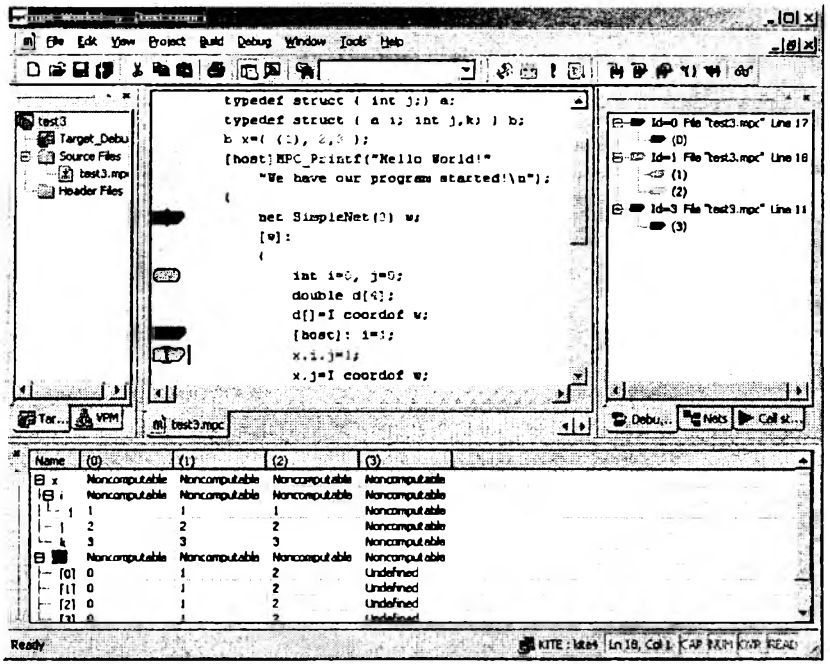


Рисунок 4. Главное окно отладчика mpC Workshop.

Для того, чтобы упростить отображение текущих позиций процессов и управление процессами в отладчике введено понятие курсора отладки. Курсор отладки соответствует группе процессов, находящихся на одной позиции исходного текста и имеющих один цвет, отображающий состояние процесса. Для наглядности использованы цвета светофора. Если курсор имеет красный цвет, это означает, что он заблокирован на коммуникации или синхронизации с другими процессами. Такой курсор не может выполнять команды отладчика и сообщать отладчику информацию о своем состоянии. Зеленый курсор готов к выполнению команд. Желтый курсор также свободен от каких-либо действий и может выполнять команды, но не выполняет команды пошаговой трассировки по программе, поскольку перекрашивание в желтый цвет выполняется пользователем для того, чтобы

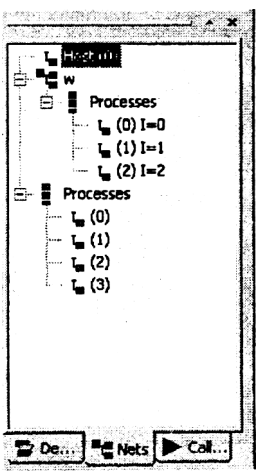


Рисунок 2. Окно вычислительного пространства.

приостановить движение определенных процессов. После каждого выполнения операторов программы курсоры автоматически перегруппируются согласно позиции в тексте, что позволяет очень удобным и масштабируемым образом управлять группами процессов, исполняющих идентичный код.

На рис. 1 справа находится окно курсоров отладки, отображающее все курсоры, их цвета и позиции в тексте программы. С помощью него можно разбивать и объединять некрасные курсоры, находящиеся в одной позиции и изменять цвет курсора с желтого на зеленый и обратно.

Как большинство интерактивных отладчиков, mpC Workshop позволяет устанавливать точки останова (breakpoints) с возможным условием срабатывания, точки изменения выражения (watchpoints), срабатывающие при изменении значения указанного выражения, и автоматически следить за значениями переменных. В нижнем окне отладчика (рис. 1) отображаются значения переменных программы. Строка таблицы соответствует значению скалярной переменной или компоненте составного выражения, а столбец – процессу. Такая схема позволяет удобно отображать распределенные значения переменных сразу на многих процессах. На рис. 1 видны значения структуры x и вектора d на процессах 0,1 и 2. Т.к. процесс 3 в данный момент заблокирован на синхронизации при создании сети, он не может передать информацию о значениях переменных.

На рис. 2 показано еще одно окно отладчика, отображающее специфичную для языка mpC иерархию вычислительного пространства. Оно отображает список всех виртуальных процессоров текущей программы и дерево сетей и подсетей, созданных в программе на текущий момент выполнения. В данном примере вычислительное пространство состоит из четырех процессов и создана сеть w, в которую вошли процессоры 0,1 и 2 имеющих в сети координату I, равную также 0,1 и 2.

Строго говоря, информация о семантической иерархии вычислительного пространства не является специфичной только для языка mpC, так, в MPI [12] аналогом подобного разбиения является множество коммуникаторов, и, хотя MPI не содержит явных средств указания иерархии коммуникаторов и групп, данная информация может быть получена путем анализа множеств процессов, входящих в коммуникаторы. Тем не менее, несмотря на то, что отображение процессов вычислительного пространства как иерархической структуры является более удобным и функциональным, нежели линейный список процессов, ни один из известных MPI отладчиков не позволяет отображать процессы указанным образом.

Для обеспечения масштабируемости отображения данных введено понятие сетевого фильтра. При большом количестве

виртуальных процессоров большинство окон оказываются перегружены ненужной информацией. Для mpC типична ситуация, когда процессоры какой-либо сети выполняют вычисления с определенными данными, в то время как на остальных процессорах значения переменных могут быть не определены или не инициализированы. В этом случае отладчик позволяет установить фильтр на отображение только компонент переменных, принадлежащих этой сети или подсети. В более сложных случаях, фильтр может объединять произвольным образом индивидуальные процессоры, сети и подсети. В некоторых случаях, таких как срабатывание точки просмотра выражения (watchpoint), на окно, отображающее данное выражение, автоматически устанавливается фильтр, содержащий только те компоненты выражения, которые были изменены во время предыдущего шага выполнения.

Использование внутренней информации mpC-программы о структуре вычислительного пространства позволяет гибко управлять отображением данных и повышает масштабируемость интерфейса в случае большого количества процессов.

Подобная организация пользовательского интерфейса удовлетворяет всем перечисленным требованиям и делает отладчик удобным и функциональным инструментом для отладки параллельных программ на языке mpC. Компромиссный подход между отображением максимального количества доступной информации и сложным конфигурированием представления данных, а также работа с высокоуровневыми конструкциями входного языка и отображение информации в терминах исходного текста программы сильно упрощают отладку параллельных приложений. Современный интерфейс и поддержка платформы Microsoft Windows делают среду разработки mpC Workshop подходящим средством для написания и отладки параллельных программ для локальных сетей персональных компьютеров, доступным для применения специалистам в прикладных областях математики и физики.

Заключение

Повсеместное распространение параллельных вычислений в настоящее время сдерживается отсутствием адекватных средств разработки параллельных приложений. Одним из основных инструментов для разработки качественного программного обеспечения является отладчик. Разработка параллельного отладчика представляет собой очень сложную задачу, в том числе с точки зрения организации пользовательского интерфейса процесса отладки.

Данная работа представляет новый подход к решению этой проблемы, реализованный в отладчике mpC Workshop, предназначенном для отладки параллельных программ на языке mpC. Предложенный подход активно использует специфику отлаживаемой программы и предоставляет удобные и функциональные средства для отладки параллельных приложений. Подход является достаточно универсальным и, с некоторыми модификациями, может быть применен для отладки других программ, в частности, использующих библиотеку передачи сообщений MPI.

Литература

1. Cherri M. Pancake, Curtis Cook, What Users Need in Parallel Tool Support: Survey Results and Analysis. Oregon State University, 1994. - 13pp.

2. Lieberman H., The Debugging Scandal and What to Do About It, Communications of the ACM, Vol. 40, No. 4, pp. 27-29. 1997.

3. Dieter Kranzlmuller, Event Graph Analysis for Debugging Massively Parallel Programs, PhD thesis, 2000, 344pp.

4. Joel Huselius, Debugging Parallel Systems: A State of the Art Report, MRTC Report no. 63, 2002, - 63pp.

5. Gregory Watson, PhD Thesis, Monash University, 2001, - 196pp.

6. Kalinov A, Ledovskikh I, The mpC parallel debugger, Proc. of PDPTA-2001, CSREA Press, 2001. - 7pp.

7.Alexey L. Lastovetsky, Parallel Computing on Heterogeneous Networks, John Wiley & Sons, 2003 - 423pp.

8. Etnus, TotalView Users Guide, Version 6.2, 2003, - 454pp.

9. Steve Sistare, Don Allen, Rich Bowker, Karen Jourdenais, Josh Simmons and Rich Title, A Scalable Debugger for Massively Parallel Message Passing Programs. Debugging and performance tuning for parallel computing systems, IEEE press, 1996 - 16pp.

10. Sun Microsystems, Prism 7.0 User's Guide, 2003, - 282pp.

11. Robert Hood, The p2d2 Project: Building a Portable Distributed Debugger. Proceedings of SPDT'96, 1996, - 10pp.

12. Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, version 1.1, 1995, - 232pp.

13. Kalinov A, Lastovetsky A, Heterogeneous Distribution of Computations While Solving Linear Algebra Problems on Networks of Heterogeneous Computers, Journal of Parallel and Distributed Computing, 61, 4, pp.520-535, 2001.

14. Leslie Lamport, Time, Clocks, and the Ordering of Events in a Distributed System, Communications of the ACM, vol 21, 1978, 8pp.

Булочникова Н. М., Сальников А. Н. Разработка прототипа CASE средства создания программ для гетерогенных многопроцессорных систем "PARUS"

На базе графа алгоритма создаётся параллельная программа, которая затем будет исполняться на гетерогенной многопроцессорной системе. Процессор, на который будет назначена вершина графа, выбирается из соображений динамики. Разрабатываемая система автоматизированного создания параллельных программ берёт на себя функции по балансировке нагрузки на процессоры многопроцессорной системы и функции по балансировке загрузки среды передачи данных. В процессе балансировки загрузки будет учтена информационная структура задачи. Для балансировки загрузки процессоров возможно также создание статического расписания исполнения графа алгоритма. В процессе разработки также находится пользовательский графический интерфейс, позволяющий создавать и редактировать граф алгоритма, а также статическое расписание.

1. Введение

В связи с широким распространением многопроцессорных систем различной архитектуры с распределённой памятью и кластеров в частности при помощи распараллеливания вычислений можно добиться высокой производительности. Однако одну и туже параллельную программу могут захотеть исполнять на многопроцессорных системах с различными архитектурами. Довольно часто в этом случае получается значительный проигрыш в производительности. Таким образом, решение комплекса проблем, которые возникают в связи с параллельным программированием, требует предварительного анализа алгоритма и, возможно, написания программы совсем в другом ключе. Система автоматизированного распараллеливания программ ***PARUS*** решает проблему создания переносимых с точки зрения эффективности программ для многопроцессорных систем. Следует отметить, что человек, который придумывает метод решения своей задачи, обычно далек от проблем программирования для многопроцессорных систем, следовательно, проблема создания автоматизированных средств проектирования параллельных приложений чрезвычайно актуальна в настоящий момент времени.

Для удобства использования и для того чтобы ***PARUS*** смог стать полноценным CASE-средством, система должна иметь достаточно

мощный и удобный в обращении графический интерфейс, который смог бы наглядно отображать как структуру самой параллельной программы, так и процессы, сопутствующие её выполнению. Тогда человек, использующий эту систему для решения своей задачи, сможет создать параллельную реализацию задачи, даже не имея навыков параллельного программирования для многопроцессорных систем.

Хочется отметить ***DVM*** - систему (Distributed Virtual Machine), по своим задачам, она в некотором смысле, близка к рассматриваемому ниже походу. Она также близка к ***HPF*** и ***OpenMP***. Язык основан на псевдокомментариях в тексте программы, которые указывают, что необходимо выполнять параллельно, а что нет, и каким образом распределять данные по многопроцессорной системе. ***DVM*** использует директивы компилятору похожие на директивы ***HPF***. Код, полученный компилятором ***DVM*** по программе с псевдокомментариями ***DVM,*** есть исходный код на ***C*** или ***FORTRAN*** с вызовами ***MPI*** функций. Полученный таким образом код параллельной программы может быть эффективно выполнен на распределённой гетерогенной многопроцессорной системе. [1]

В Массачусетском Технологическом Университете (MIT) разработан язык параллельного программирования ***Cilk***. ***Cilk*** – язык для параллельного многопоточного программирования, основанный на ANSI ***C***. ***Cilk*** спроектирован для написания параллельных программ общего назначения, но особенно эффективен для тех задач, которые трудно написать в стиле передачи сообщений. Программная компонента языка во время запуска программы позволяет предсказывать момент окончания выполнения программы. Философия ***Cilk***, заключается в том, что человек концентрирует своё внимание на структуре программы, выражении параллелизма. Данный язык содержит C с тремя добавочными ключевыми словами, которые описывают параллелизм и синхронизацию. Компилятор ***Cilk*** преобразует программу в программу на языке ***C*** с использованием вызовов ***p_thread***. [2]

2. Постановка задачи

Алгоритм решаемой задачи, представляется в виде ориентированного графа, где в вершинах сосредоточенны действия над данными, а рёбра символизируют зависимость по данным. При этом дуга направлена от вершины источника данных, к вершине принимающей данные. Следует отметить, что данные операции вовсе не обязательно являются простейшими элементарными операциями, и система в первую очередь предназначена для работы в случае, когда вершины графа алгоритма являются полновесными, тяжёлыми операциями, возможно даже целыми программами, для которых

зачастую бывает сложно оценить время исполнения на одном процессоре. В графе не должно быть ориентированных циклов. Граф выстраивается в ярусную форму, где каждый уровень получает данные от предыдущего уровня. Вершины графа на каждом уровне независимы между собой. Вершины графа с большего уровня используют данные, полученные в процессе выполнения действий определённых в вершине графа с меньшего уровня.

Под расписанием будем понимать желаемый порядок вызовов узлов графа алгоритма на процессорах. Расписание должно быть допустимым: 2 узла графа не могут одновременно выполнятся на одном процессоре; должен соблюдаться порядок приема и передачи данных. Также расписание должно быть минимальным по времени исполнения программы или близким к минимальному.

Описание самого графа и расписания для него содержится в текстовых файлах, которые можно подать на вход набору утилит. Подробное описание утилит можно найти в статье [3], а информацию о форматах файлов в статье [4].

В системе имеются несколько утилит: утилиты, которые меряют производительность процессоров и каналов связи, утилита, строящая статическое расписание по графу заданному на вход, а также информации о загрузке каналов связи, утилита, преобразующая граф программу в исходный код программы написанной на ***C***++ со встроенными в неё ***MPI*** вызовами. В системе есть утилита, отображающая граф алгоритма на экране (без возможностей редактирования оного), предназначенная для поиска некорректных мест в текстовом файле для созданного графа алгоритма. Одна из самых привлекательных утилит – утилита по преобразованию исходного текста ***C*** программы в граф алгоритма. К сожалению, ограничения на ***C*** программу весьма существенные, что приводит к тому, что, несмотря на всю привлекательность данной утилиты, воспользоваться ей можно в достаточно редких случаях. Данная утилита обсуждена в статье [3], а проблемы представления последовательных программ в виде графа и поиск параллелизма широко обсуждены Воеводиным В.В. в работе [5].

Далее обсудим утилиту, позволяющую в графическом виде редактировать граф алгоритма и расписание.

3. Редактор графа

Редактор позволяет создавать новый граф, редактировать существующий, а также изменять расписание выполнения графа программы на многопроцессорной системе. Редактор реализован на языке ***Java2*** из соображений независимости от операционной системы. В проекте была использована компонента ***JGraph***, которая была взята с одноименного открытого проекта.

Граф представляется на экране как набор прямоугольников соединённых стрелками, стрелки зависимость по данным между вершинами. Чем темнее цвет прямоугольника, тем больший вес сопоставлен соответствующей ему вершине графа, и тем более он трудоёмок в вычислительном смысле.

Рассмотрим подробнее работу с нодами и ребрами. На вершину графа можно открыть форма, представленную на рисунке 1, в которой пользователь может изменить атрибуты вершины графа.

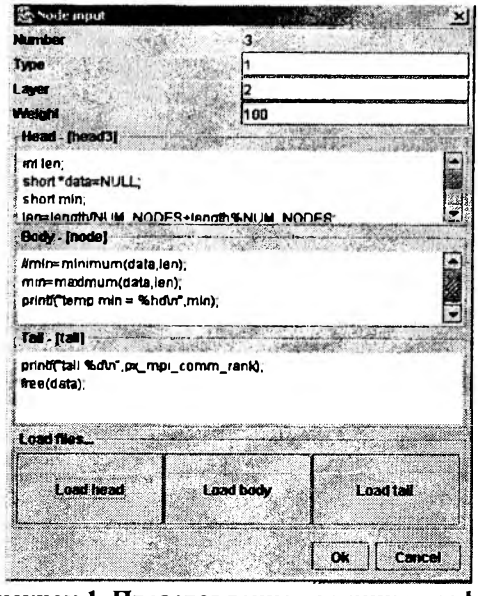


Рисунок 1. Представление вершины графа.

К атрибутам вершины относятся: номер, уровень в графе, вес, тип, а также программный код, выполняемый в вершине. Изменение текста создаваемой граф программы можно произвести вручную, отредактировав соответствующий текстовый файл. Аналогичные действия можно провести с ребрами. Здесь атрибутами ребер будут его

номер, тип, вес, число массивов посылаемых данных и сами массивы, нарезанные на чанки. (рис. 2.)

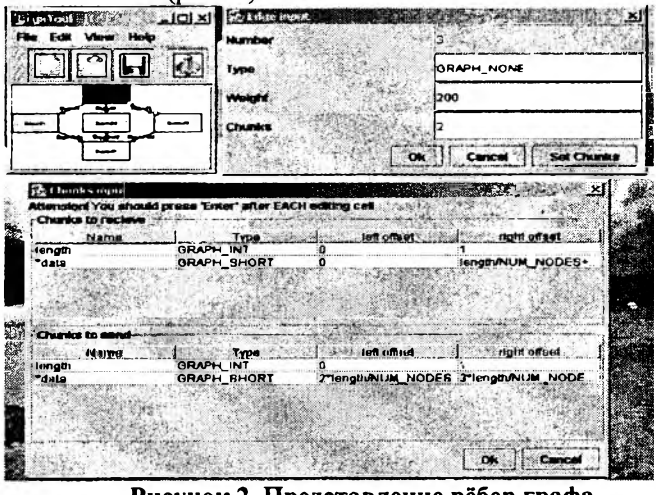


Рисунок 2. Представление рёбер графа.

Возможно, также отредактировать файл с расписанием. Таких файлов может быть несколько. Пользователь должен выбрать нужный ему файл. Визуализация расписания производится по аналогии с визуализацией графа. Пример расписания представлен на рисунке 3.

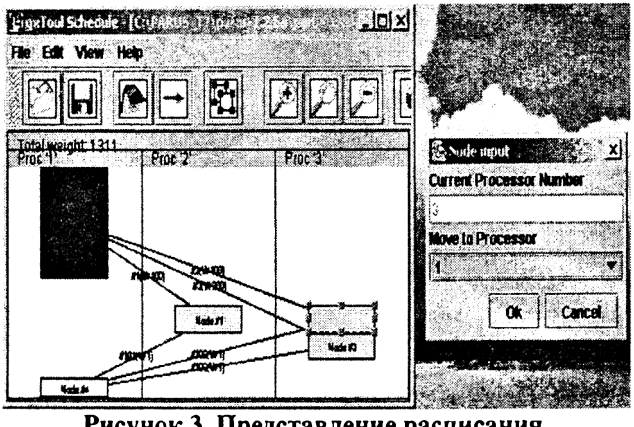


Рисунок 3. Представление расписания.

Вертикальные разделения окна обозначают процессоры, на которые распределены вершины графа. Временем выполнения каждой вершины на каком-либо процессоре считаем вес этой вершины. Редактором считается максимальное из времён выполнения вершин на

процессорах - это есть предположительное относительное время выполнения программы. Имеются возможности просмотреть текст файла с описанием расписания, переносить вершины графа на другой процессор, уменьшать/увеличивать изображение, включить режим просмотра расписания без ребер.

4. Результаты

Система протестирована на нескольких примерах: от простых, до сложных. Были предложены следующие тесты:

- задача поиска экстремума в wav-файле;
- задача перемножения матриц;
- задача построения многослойного персептрона (узел графа есть один нейрон, дуги - синапсы и аксоны);
- задача вычисления скалярного произведения 2-х векторов большого размера.

Минимальная сложность графа составила 3 вершины, максимальная – 1000 вершин. По рёбрам – максимальное число рёбер 4500.

5. Заключение

Разработка данной системы производится при поддержке грантов РФФИ 02-07-90130 и 03-07-06108. Приносим благодарности чл.-кор. ран Королёву Л.Н. и дфмн. Поповой Н.Н. за консультации в процессе разработки, а также Цветкову Т.Н. за помощь в написании тестовых примеров.

Литература

1. Коновалов Н.А., Крюков В.А., Сазанов Ю.Л.. "C-DVM - язык разработки мобильных параллельных программ". Программирование N 1, 1999. стр. 54-65.

2. Supercomputing Technologies Group MIT Laboratory for Computer Science "Cilk 5.3.2 Reference Manual" 2001. стр. 1-42.

3. Сазонов А.Н., Сальников А.Н., Кареев М.В. "Прототип системы разработки параллельных программ для гетерогенных многопроцессорных систем." Тематический сборник факультета ВМиК МГУ "программные системы и инструменты" N3 2002г. стр. 139-151.

4. Сальников А.Н. “Некоторые технические аспекты инструментальной системы для динамической балансировки загрузки процессоров и каналов связи.” Тематический сборник факультета ВМиК МГУ “программные системы и инструменты“ N3 2002г. стр. 152-164.

5. Воеводин В.В. “Информационная структура алгоритмов”. - М.: Изд-во МГУ, 1997г. 139 стр.

Аннотации

Ключников В.О., Костенко В.А., Маркин М.И. Анализ эффективности нейросетевых, регрессионных и спектральных методов анализа временных рядов. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

Проводится сравнительнение нейросетевых, регрессионных и спектральных методов анализа временных рядов по критериям: использование априорных сведений о процессе, породившем временной ряд, потенциально достижимая точность построенной модели ряда, возможность использования разнородной входной информации, вычислительная сложность, возможность выявления различных компонент ряда.

Библиогр.: 9 назв.

Сухомлина А.И. Метод построения корректной политики безопасности для систем электронного документооборота. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

В работе рассматривается математическая модель формальной семантики механизмов разграничения доступа в системах электронного документооборота. Для данной модели определены критерии корректности механизмов доступа и построенных на их основе политик безопасности, а также предложены алгоритмы проверки корректности механизмов и политик безопасности. Рассмотренный подход успешно апробирован на практике при создании систем документооборота производственного назначения.

Ил.: 1

Библиогр.: 6 назв.

Брусенцов Н.П. Парадоксы логики, здравый смысл и диалектический постулат Гераклита-Аристотеля. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

Простое и исчерпывающее решение проблемы парадоксальности импликаций путем представления содержательного следования в трехзначной алгебре логики как нечеткой совокупности идивидных конъюнкций. Диалектическое на основе постулата Гераклита-Аристотеля воссоздание силлогистики.

Доложено на Ломоносовских чтениях 2003г. на факультете ВМиК МГУ.

Библиогр.:6 назв.

Нековаль С.П. Метод фазовой корреляции решения задачи анализа движения в видеоизображениях. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

Метод фазовой корреляции – один из самых эффективных на сегодняшний день методов анализа движущихся объектов в видеоизображении. В статье описываются основные этапы реализации метода и приводится краткое математическое обоснование его работы. Рассматриваются различные способы оптимизации алгоритма и повышения качества результата. Приводится характеристика эффективности по сравнению с традиционным переборным алгоритмом.

Ил.: 9

Библиогр.: 7 назв.

Подшивалов А.Ю. Определение пространственного расположения 3D объектов по маркерам. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

Работа посвящена проблеме создания алгоритмов определения пространственного расположения объектов, имеющих большие размеры и протяженность вдоль нескольких произвольно ориентированных осей (таких как современные и перспективные долговременные орбитальные станции и их сегменты). Данную задачу необходимо решать на каждом шаге работы системы моделирования. 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 точности вычислений в связи с высоким риском столкновений объектов. В работе предлагается подход с использованием высокоточных маркеров, привязанных к объектам. При этом в систему моделирования поступает два вида информации: априорная информации о поверхностях объектов, и апостериорная - о координатах маркеров. Показано, что такой подход позволяет эффективно решать задачу для 3D-объектов высокой сложности.

Библиогр.: 8 назв.

Зленко П. А. Интегрированная система паралингвистической обработки голоса. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

Статья посвящена рассмотрению вопросов построения систем обработки голоса, учитывающих звуковое паралингвистическое содержание речи. Приведён краткий обзор подобных систем и предложена концепция разработки интерактивной среды, предоставляющей новые возможности - нормализация дикторской речи и подмена индивидуальности голоса.

Библиогр.: 14 назв.

Саак А.Э. Анализ функционирования многопроцессорных систем коллективного пользования. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

В работе анализируется взаимодействие сред поставляющих и потребляющих компьютерный сервис. Изучается случайная величина, отвечающая среде, поставляющей компьютерный сервис, на основе которой определяется воспринимательная способность многопроцессорной системы, характеризующая возможности по обслуживанию новых пользователей. Описывается случайная величина, характеризующая среду, потребляющую компьютерный сервис. Совокупность данных случайных величин позволяет значительно полнее представить функционирование многопроцессорной системы с множеством пользователей.

Ил.: 5

Библиогр.: 5 назв.

Веселов Н.А. Алгоритм расчёта средней задержки пакета в сетях передачи данных типа wormhole. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

Статья посвящена исследованию сетей передачи данных типа wormhole (Myrinet, Servernet II, Sunfinity), которые применяются при построении кластерных вычислительных комплексов, а также специализированных систем реального времени. В работе представлен алгоритма расчёта важнейшей характеристики сетей передачи – средней задержки пакета для данного класса сетей.

Библиогр.: 12 назв.

Ил.: 5

Библиогр.: 12 назв.

Кичигин Д.Ю., Машечкин И.В, Петровский М.И. Методы планирования потока инструкций для конвейерных RISC архитектур. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

Планирование инструкций – это семейство программных и аппаратных техник, направленных на повышение скорости выполнения потока команд за счет параллельного выполнения отдельных машинных операций. Существуют динамические (выполняются аппаратурой во время выполнения программы) и статические(выполняются на стадии компиляции программы) техники планирования.

В статье представлен обзор алгоритмов статического планирования инструкций. Отражены особенности алгоритмов планирования для конвейерных RISC архитектур.

Библиогр.: 7 назв.

Корухова Л. С., Малышко В. В. Ассоциативные и дедуктивные методы в планировании многошаговых многовариантных задач. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

В работе рассматривается система планирования решений сложных задач, реализующая использование различных стратегий и методов построения планов. Описан способ организации процесса планирования на основе управляющих стереотипов. При этом управляющие стереотипы в зависимости от состояния процесса построения плана реализуют выбор конкретных стратегий планирования. Приведены примеры построения планов с использованием различных методов планирования: ассоциативного планирования, прямой и обратной цепочек рассуждений.

Ил.: 6

Библиогр.: 5 назв.

Кичигин Д.Ю., Машечкин И.В., Петровский М.И. О применении подходов Data Mining для обнаружения вторжений. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

В настоящей статье рассматривается применение методов Data Mining в системах обнаружения компьютерных вторжений. В статье рассматриваются основные подходы, применяемые для решения задачи обнаружения вторжений, показываются достоинства и недостатки этих подходов. Далее дается краткий обзор задач, где традиционно используются методы Data Mining, и рассматривается их применение для задачи обнаружения вторжений.

Библиогр.: 16 назв.

Герасимов С.В. Алгоритмы поиска ассоциативных правил. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

В работе рассматриваются одна из задач Data Mining - поиск ассоциативных зависимостей в базах данных большого объема. Проведено сравнение двух алгоритмов выявления булевых ассоциативных правил: Apriori и FP-дерево.

Библиогр.: 6 назв.

Петровский М. И., Сорокина Д. В. Адаптация алгоритма классификации на основе нечетких деревьев решений для анализа многомерных данных. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

В этой работе пойдет речь о применении алгоритмов классификации для многомерной модели представления данных на основе n-мерного информационного куба. Такое представление характерно для хранилищ, использующих технологию OLAP (on-line analytical processing). Существует много алгоритмов интеллектуального анализа данных (Data Mining), применимых для реляционного представления информации, но они, как правило, показывают плохие результаты для многомерной модели представления данных. Отчасти это происходит из-за таких особенностей многомерных структур, как разреженность. Поэтому в этой области еще предстоит разработать новые методы анализа данных, которые бы могли эффективно работать именно с разреженными многомерными кубами. Один из таких методов предложен в этой работе.

Ил.: 7

Библиогр.: 7 назв.

Леонов М.В., Новиков М.Д., Казакова Н.Л., Леонов В.М. Программный инструментарий для интеграции и обработки библиографических данных в таксономических исследованиях. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

Описан способ интеграции библиографических данных из нескольких баз данных и других источников в единую библиографическую информационно-поисковую систему с возможностями вывода результатов запросов в различных форматах с применением XML- и XSLT-технологии.

Библиогр.: 4 назв.

Брусенцов Н.П., Владимирова Ю.С. Логико-алгебраический процессор. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

Программно реализованный на основе ДССП алебраический процессор с функционально полным набором операций над переменными типа «n-арное булево выражение», кодируемыми векторами битов или тритов. Процессор позволил компьютеризировать решение булевых уравнений, преобразование булевых выражений в заданную форму и прочие алгебраические процедуры.

Доложено на Ломоносовских чтениях 2003г. на факультете ВМиК МГУ.

Библиогр.: 3 назв.

Бурцев А.А., Рамиль Альварес Х. Средства объектно-ориентированного программирования в ДССП . // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

Рассматриваются некоторые аспекты модификации новой версии ДССП, ядро которой разрабатывалось на языке Си, в целях реализации для нее средств объектно-ориентированного программирования.

Доложено на Ломоносовских чтениях на факультете ВМиК МГУ 24 апреля 2003 г.

Ил.: 1

Библиогр.: 3 назв.

Маслов С.П., Рамиль А. Х., Сидоров С.А. Реализация МСО "Наставник" на микрокалькуляторе МК-85. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

Обосновывается перспективность развития системы обучения "Наставник" с ориентацией на использование автономных, т.е. не нуждающихся в постоянной связи с центральным компьютером, терминалов. Обсуждается специфика и способы реализации терминала путем программной модификации существующих тиражируемых устройств. Приводятся краткие сведения об отечественном калькуляторе "Электроника МК-85" и его штатном ПО. Описываются процесс формирования модифицированного ПО и используемые для этого средства.

Доложено на Ломоносовских чтениях 2003г. на факультете ВМиК МГУ.

Библиогр.: 8 назв.

Акишин Р.С. Построение индикатора рынка на основе нейросетевого подхода. Использование однородной Марковской модели для анализа выходов нейросети в задачах классификации. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

Статья посвящена вопросам применения однородной Марковской модели для анализа выходов нейросети в задачах классификации на основе временных рядов. В качестве модели в статье предлагается рассмотреть троичный индикатор рынка, и на его примере изучаются проблемы выбора архитектуры нейросети, обучения, кодирования выходов и анализа результата полученного нейросетью.

Ил.: 4

Библиогр.: 6 назв.

Карганов К.А. Организация интерфейса отладчика для параллельных программ. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

Статья посвящена проблеме организации пользовательского интерфейса отладчика для параллельных программ. В ней дается обзор основных типов отладчиков и освещаются основные проблемы, возникающих при разработке пользовательского интерфейса параллельного отладчика. На примере известных отладчиков для параллельных программ описываются существующие подходы к организации пользовательского интерфейса. В качестве основного результата описывается подход, реализованный в отладчике mpC Workshop для среды программирования mpC и проводится сравнение выбранного подхода с уже существующими.

Ил.: 2

Библиогр.: 14 назв.

Булочникова Н. М.,Сальников А. Н. Разработка прототипа CASE средства создания программ для гетерогенных многопроцессорных систем "PARUS".// Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2003.

Система автоматизированного распараллеливания программ ***PARUS*** решает проблему создания переносимых с точки зрения эффективности программ для многопроцессорных систем. Для удобства использования и для того чтобы ***PARUS*** смог стать полноценным CASE-средством, система должна иметь достаточно

мощный и удобный в обращении графический интерфейс, который смог бы наглядно отображать как структуру самой параллельной программы, так и процессы, сопутствующие её выполнению. Тогда человек, использующий эту систему для решения своей задачи, сможет создать параллельную реализацию задачи, даже не имея навыков параллельного программирования для многопроцессорных систем.

Ил.:3

Библиогр.: 5 назв.

Для заметок

Для заметок

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 4

Под общей редакцией
чл.-корр. РАН 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 25.12.2003 г.
Формат 60x90 1/16. Усл.печ.л. 13,75. Тираж 200 экз. Заказ 058.
Тел. 939-3890, 939-3891, 928-1042. Тел./Факс 939-3891.
119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова,
2-й учебный корпус, 627 к.