САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ЛАВРОВСКИЕ ЧТЕНИЯ 2013

МАТЕРИАЛЫ ВСЕРОССИЙСКОЙ НАУЧНОЙ КОНФЕРЕНЦИИ ПО ПРОБЛЕМАМ ИНФОРМАТИКИ

ПЛЕНАРНЫЕ ЗАСЕДАНИЯ

23–26 апреля 2013 г.,
Санкт-Петербург

Санкт-Петербург
«ИПК «БЕРЕСТА»
2013

УДК 004(063)
ББК 32.97я431
Л18

Печатается по рекомендации
кафедры системного программирования
Санкт-Петербургского государственного университета

Л18 **Лавровские** чтения 2013: Материалы всероссийской научной конференции по проблемам информатики. Пленарные заседания. 23–26 апр. 2013 г., Санкт-Петербург. — СПб.: ООО «ИПК «Береста», 2013 — 792 с.
ISBN 978-5-906670-02-1

Пленарные заседания научной конференции по проблемам информатики Лавровские чтения проводятся в дни всероссийской конференции СПИСОК с целью организации научного общения преподавателей и молодых исследователей с известными российскими и зарубежными учёными в области информатики.

Для студентов и аспирантов естественно-научных специальностей.

УДК 004(063)
ББК 32.97я431

ISBN 978-5-906670-02-1

© Санкт-Петербургский государственный университет, 2013

ХОРОШАЯ ТЕХНОЛОГИЯ ДЕЛАЕТ ТРУДНУЮ ЗАДАЧУ ПРОСТОЙ

Терехов А. Н.

д.ф.-м.н., профессор, зав. каф. системного программирования СПбГУ
директор НИИ Информационных технологий СПбГУ
председатель программного комитета, руководитель секции

Введение

Давно известно, что новые, неизвестные ранее методы счёта, решения каких-то уравнений, других сложных задач чаще всего появляются при появлении новых технологий. Самый простой пример — умение считать. И в Средние века люди, умеющие хорошо считать, высоко ценились. Например, известно, что такой великий математик, как Эйлер очень хорошо умел считать и использовал это умение для решения самых разных задач. Потом, когда появились логарифмические линейки, арифмометры, ЭВМ, в конце концов, умение считать перестало считаться уровнем квалификации математика-расчетчика, но все равно — умение прикинуть результат, быстро понять, дает ли ЭВМ правильный ответ или в алгоритме есть ошибка — и сейчас остается важным.

В последнее время появилось много примеров, когда наличие или, наоборот, отсутствие определенной технологии принципиально меняет ситуацию. Начнем с наиболее близкого нам примера. Появление в 50–60-ых годах алгоритмических языков высокого уровня Фортран, Алгол 60, Кобол и других позволило существенно поднять производительность труда программистов, но не было бы возможно без появления технологий анализа исходных текстов на этих языках, оптимизации, генерации объектного кода и других трансляторных техник.

Еще один пример — давно известно, что если реализовать микропрограмму для выполнения какой-то часто встречающейся и «времяпожирающей» функции, то можно выиграть во времени счёта в 10–20 раз. Все машины Ряда 2, выпускаемые в СССР (это ЕС ЭВМ, номера серий которых оканчивались на 5), имели возможность динамического микропрограммирования, но никто этой возможностью не пользовался, поскольку написать микропрограмму для «прообраза» ЕС ЭВМ IBM Mainframe — это тяжелая работа и никто не хотел этим заниматься, хотя все понимали, что можно выиграть во времени счёта. В середине 80-ых годов один из сотрудников лаборатории системного программирования мат-мех факультета СПбГУ Николай Фоминых придумал способ микропрограммирования на языках высокого уровня [1] и, как по мановению волшебной палочки, эта задача из очень сложной превратилась в задачу, которую мы даже не хотели принимать в качестве дипломной работы — это стало задачей уровня курсовой работы студента 3–4 курса.

Ровно то же самое происходит сейчас с графическими языками программирования. Трудно заставить проектировщика использовать только стандартный UML, это тяжелая работа, и даже правильно понять, где и какую из 13 типов диаграмм UML нужно использовать, не каждый может, а уж сделать хороший проект (не картинку для начальников, а проект, который автоматически можно превратить в исполняемую программу для ЭВМ) — это работа очень сложная. И здесь со временем нашелся выход — программисты стали сочинять и применять специфические для каждой предметной области языки Domain Specific Languages (DSL). И опять оказалось, что очень трудные задачи с использованием хорошо подобранных DSL для каждой предметной области превращаются в простые.

Понятно, что для каждого DSL разрабатывать свои графические редакторы, репозитории, генераторы кода, средства отладки и т. д. очень дорого. И здесь нашлось решение — метатехнологии, то есть технологии, которые по некоторому набору не слишком сложных формальных описаний способны автоматически сгенерировать технологию для нужного DSL [2].

Постановка задачи

Этот доклад посвящен способу проектирования кристаллов. При проектировании кристалла (мы будем говорить о гибких кристаллах FPGA — Field Programmable Gate Array) алгоритм работы кристалла принципиально отличается от тех традиционных алгоритмов, которые программисты каждый день пишут на C или подобных языках. Принято считать, что выигрыш во времени работы с использованием программируемых кристаллов достигается за счет распараллеливания. В кристалле можно запустить одновременно 100–200 действий, процессов и за счёт этого существенно выиграть во времени исполнения по сравнению с традиционным программированием, но наш опыт показал, что основной выигрыш достигается вовсе не за счет прямого распараллеливания, а, в основном, за счёт конвейеризации. Многие действия невозможно выполнить за один такт, поэтому их приходится выполнять в течение нескольких последовательных тактов. Если при этом используются разные устройства, то всегда выгодно устроить так называемый конвейер — последовательность действий, в которой на первом такте какой-то кусок счета выполняется группой устройств, промежуточные результаты передаются другой группе устройств, которые в следующем такте будут продолжать эту работу, но первая группа устройств в следующем такте также может взять другую порцию данных и выполнить свою работу, причем глубина этого конвейера может быть довольно большой. Это общеизвестная истина, и все это знают, но реально при программировании кристаллов большие и глубокие конвейеры реализовать очень тяжело.

Традиционными языками программирования для проектирования кристаллов являются VHDL и Verilog, в которых специальных средств органи-

зации конвейера нет. Программист, пишущий, скажем, на VHDL, должен сам продумать систему передачи данных через какие-то рабочие регистры, сам обеспечить синхронизацию (если нужная порция данных не пришла, то конвейер должен приостановиться, а потом, в момент появления нужных данных, возобновить свою работу). И все это настолько хлопотно и сложно, что глубоких конвейеров обычные программисты на VHDL не создают. Конечно, на все случаи жизни есть исключения. Если, например, проектируется кристалл массового процессора, то можно себе позволить затратить огромные человеческие усилия для того, чтобы спроектировать глубокий конвейер, поскольку потом этот процессор будет использоваться много лет в самых разных задачах и поэтому усилия, несомненно, окупятся. Но ведь далеко не всегда создается именно процессор общего применения — самые массовые работы выполняются для каких-то узкоспециальных задач, которые тоже времяемкие (англ. Time-consuming, «времяпожирающие» задачи). Но здесь уже люди мирятся с недостатками, недостаточно высокой эффективностью, здесь действует другой принцип — time to market, время выхода на рынок (пусть кристалл будет несколько хуже, чем мог бы быть, но главное выпустить кристалл с нужными характеристиками на рынок раньше конкурентов).

Основным мотивом этого доклада явилось появление на рынке продукта фирма Xilinx Vivado [3], который рекламируется как автоматический конвертор с языка C в VHDL и Verilog. При внимательном рассмотрении видно, что реальные ограничения на входные программы, которые способны быть конвертированы в VHDL, очень высоки — никакой динамики, циклов и массивов с динамически вычисляемыми границами. Всё рассчитано на то, что сегодня в больших ЭВМ с большой памятью можно построить синтаксическое дерево разбора, построить графы передачи данных и графы передачи управления, на них провести необходимые оптимизации и, когда такие графы статичны (полностью известны во время трансляции), можно надеяться на применение различных оптимизационных механизмов и, как утверждают авторы средства, на практически автоматическое получение приличных VHDL программ для кристаллов. Сейчас ещё трудно оценить реальные возможности этого средства, насколько оно реально эффективно.

Не лучше ли пойти другим путём, то есть дать программисту удобные выразительные средства и мощную эффективную технологию, тогда можно будет решать существенно более широкий класс задач?

Нетривиальный пример

Наш коллектив уже много лет работает над построением специализированного инструментального средства проектирования аппаратных реализаций сложных приложений на базе разработанного нами языка HaSCoL (Hardware and Software CoDesign Language) [4, 5, 6, 7, 8]. В этом языке есть

удобное для программиста средство задания явного параллелизма — несколько действий можно выписать через разделитель «|», тем самым транслятору дается указание, что эти действия должны выполняться параллельно в течение одного такта, и средство явного задания конвейера — несколько действий, разделенных символом «;» задают конвейер, причем система берет на себя задание регистров, хранящих промежуточные результаты, синхронизацию, взаимодействие конвейеров, их приостановку и возобновление. Мы верим, что языки без явного параллелизма и явного задания конвейера не могут быть столь эффективны, как языки, в которых таковые действия предопределены, и надеемся показать преимущества нашей платформы на реальном производственном примере.

В качестве такого примера мы выбрали поиск изображения в кадре. Чтобы не мучать читателя тонкостями алгоритма мы выбрали стандартный корреляционный алгоритм, предполагая, что есть экран 512 на 512 пикселей, в котором надо найти изображение (эталон) размером 128 на 128 пикселей. Это довольно времяемкий алгоритм, но хорошо работающий. В двух словах опишем суть алгоритма. Просматривается исходный экран, для каждого фрагмента экрана размером 128×128 происходит его нормировка, а затем вычисляется скалярное произведение получившихся нормированных значений пикселей с нормированным значением эталона. Максимум этих скалярных произведений указывает на найденный фрагмент.

Обозначим размеры экрана M∗N (в данном примере $M = N = 512$). В этом изображении надо найти эталон — массив из пикселей размером $m * n$ (в данном примере $m = n = 128$).

Нормировка осуществляется по следующей формуле:

$$a'_{ij} = \frac{a_{ij} - \overline{a}}{\sqrt{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (a_{ij} - \overline{a})^2}},$$

где $\overline{a}$ — среднее из всех a_{ij} фрагмента экрана. В предложенной схеме счета важную роль играет сумма

$$\sum_i \sum_j (a_{ij} - \overline{a})^2,$$

Нетрудно показать, что

$$\sum_i \sum_j (a_{ij} - \overline{a})^2 = \sum \sum a_{ij}^2 - m * n * \overline{a}^2.$$

Такой вид намного удобнее для счёта, потому что можно сразу в процессе ввода данных копить суммы значений пикселей и их квадратов, а в первом варианте используется среднее, которое в исходном виде неизвестно.

Поскольку мы планируем финальный результат получить в виде аппаратной реализации в кристалле, вычисления мы проведём с целыми числами (в том числе и вычисление корня), иногда прибегая к масштабированию.

Оценим сложность этого алгоритма. Фрагментов экрана, подлежащих нормировке и сравнению с эталоном будет $(512-128)*(512-128)=384*384$.

В каждом фрагменте нужно найти сумму элементов для вычисления среднего и сумму их квадратов, то есть выполнить $2*128*128$ действий, чтобы уменьшить количество сложений, мы применяем стандартный для такого рода задач прием, когда суммы по строкам хранятся «нарастающим итогом», то есть получается массив, в котором каждый элемент является суммой n предыдущих элементов. Каждый новый элемент строки получается сложением с предыдущим и вычитанием элемента, отстоящего на *n* элементов назад.

Таким образом, для получения суммы всех элементов очередного фрагмента надо всего $2*128$ действий (сложить элементы последнего столбца), а не $2*128*128$.

Для вычисления квадратного корня из 32-битового целого числа, удалось найти алгоритм с циклом на 16 повторений несложных действий (5–7 действий).

Для нормировки элементов и получения скалярного произведения уже никаких упрощений не применялось, таким образом, внутри цикла $384*384$ возникает цикл $128*128$ с примерно 20 действиями внутри.

Таким образом, итоговую сложность этого алгоритма можно оценить в $384*384*(2*128+6*16+10*128*128)=384*384*164\,192$ действий.

Программа на языке C по этому алгоритму на MacAir работает около 30 сек. По моей просьбе аспирант Станислав Сартасов, используя систему CUDA, применил GPU, где распараллелил внутренний цикл на 128. Он получил время счета порядка 0,5 секунд на хорошем процессоре с 2 ядрами.

Предлагаемое решение

Перейдём к описанию программы (основные элементы программы приведены в конце доклада), используя идентификаторы из её кода на языке HaSCoL (*s*, *s*2, *az* и так далее). Вначале вводим массив *a* (эталон) и считаем сумму всех элементов и сумму квадратов элементов

$$zv=\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}a_{ij},\ s2=\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}a_{ij}^2.$$

Затем считаем знаменатель нормировки эталона

$$az=\sqrt{s2-m*n*\left(\frac{zv}{m*n}\right)^2}=\sqrt{s2-\frac{zv^2}{m*n}}.$$

Теперь заменяем значения элементов эталона их нормированными значениями

$$ap_{ij}=\frac{a_{ij}-\frac{zv}{m*n}}{az}=\frac{m*n*a_{ij}-zv}{m*n*az}.$$

Если a_{ij} занимает 8 бит без знака, то $\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} a_{ij}$ займёт 22 бита, a_{ij}^2 займет 16 бит, а $\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} a_{ij}^2$ — 30 бит (напоминаем, $m=n=128$).

Аналогичные расчеты выполняются над всеми прямоугольниками экрана размером $m*n$, верхний левый угол которых определяется индексами k и l, в каждом из этих прямоугольников вычисляется $zp_{k+I,I+j}$ аналогично ap_{ij}, причем $k=0\ldots M-m+1$, $l=0\ldots N-n+1$.

Итоговое значение расчетной функции для прямоугольника k, l

$$F_{kl}=\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} zp_{k+i,l+j}*ap_{ij}.$$

Нас интересует максимум этой функции и индексы k и l, где он достигается.

В знаменателе получатся линейные сомножители $m*n$, которые на сравнения не повлияют, поэтому их учитывать не будем.

Суммы по строкам собираются в массиве

$s2\ [m,N]$ 15 бит

суммы квадратов по строкам собираются в массиве

$s2\ [m,N]$ 23 бит.

Все массивы в кристалле хранятся в памятях bram, каждая из которых допускает не более двух операций чтения за такт.

Сначала продемонстрируем идею организации конвейера на простом примере, помеченном скобкой 1 на листинге. В $m-1$ строке (счет с 0) добрались до $n-1$ столбца, таким образом образовался первый фрагмент для счета – прямоугольник, левый верхний угол которого имеет координаты $k=0$, $l=0$. Для получения суммы всех элементов этого прямоугольника надо сложить элементы до последнего столбца (где уже есть суммы по строкам). Как мы уже увидели, этот фрагмент не самый критичный, поэтому тратить на него много аппаратных ресурсов не хотелось бы.

Специально для такого случая, когда автор программы считает, что нет смысла распараллеливать какое-то не очень важное действие на большом количестве элементов микросхемы, в языке HaSCoL предусмотрен оператор while и еще несколько подобных операторов. Это обычный цикл, который повторяется несколько раз на одном и том же фрагменте аппаратуры. Оператор while тормозит текущий конвейер, например, до фрагмента, помеченного скобкой 1, алгоритм выдавал по одному промежуточному результату

в такт, но с момента начала работы этого фрагмента будет выдавать один результат в 128 тактов. Зато сам цикл while, будучи один раз запущенным, будет работать до конца, не требуя дополнительной информации.

Из-за аппаратных ограничений результат чтения из такой памяти нельзя использовать в том же такте. Получается, что сложность этого фрагмента $128 * 2$ тактов. Поступим иначе: при каждом повторе цикла наращиваем счетчик и запускаем подпроцесс sum1, причем запуск подпроцесса и исполнение его первого действия умещаются в одном такте.

Подпроцесс sum1 (скобка 5) запускает конвейер из 2 стадий (чтение суммы и суммы квадратов и их добавление). Таким образом сложность $128 * 2$ заменяется на 128+1 тактов!

Далее идёт вычисление квадратного корня (скобка 2). Этот цикл while на 16 повторений запускается каждый раз, когда заканчивает свою работу предыдущий фрагмент. Это тоже конвейер — пока вычисляется квадратный корень из результата первого элемента конвейера, параллельно с ним вычисляется первый цикл while для второго элемента и так далее. Таким образом, временем счета этого участка можно пренебречь.

В скобке 3 вычисляются нормализованные элементы эталона, это разовое действие, и к тому же выполняющееся на фоне ввода первых $m * N$ элементов экрана, поэтому мы этот фрагмент обсуждать не будем.

Теперь обсудим самый интересный и времяёмкий фрагмент программы, обозначенный скобками 4 и 6. Здесь мы используем макрогенератор, реализованный много лет назад сотрудником нашей лаборатории Антоном Москалем [9]. Идея состоит в замене линейного сложения 128 элементов столбца логарифмическим сложением за 7 итераций. Но даже эти 7 итераций мы организуем в виде конвейера: на 1 шаге конвейера складываем соседнем элементы, на 2 шаге — суммы пар, потом четверок и так далее. Нам понадобится 127 сумматоров ($64 + 32 + 16 + \ldots$), но микросхемы теперь мощнее, так что это не проблема.

Таким образом, в скобке 4 описан цикл на 128 повторений, тело цикла занимает ровно 1 такт, при этом запускается 128 параллельно работающих конвейера глубиной 10 (я просто подсчитал количество символов «;» в порожденной программе).

Вообще-то после скобки 4 нужно было бы поставить 10 операторов skip (пустой оператор), как появился один skip после скобки 1, но скобкой 4 тело внутреннего цикла заканчивается, поэтому оставшиеся действия последних повторений цикла скобки 4 выполнятся уже на фоне вычислений в следующих прямоугольниках.

Таким образом, тело внешнего цикла с числом повторений $384 * 384$ содержит 3 участка — 128, 96 и 128 тактов, организованных в виде конвейера. Они практически полностью работают параллельно за исключением маленького «хвостового» участка, работающего уже после завершения внешнего цикла. Это означает, что сложность тела внешнего цикла составляет

примерно 128 тактов, а сложность всей программы $384 * 384 * 128$ плюс небольшая константа, что более чем в 1 200 раз меньше исходного варианта!

На наш взгляд, реализовать столь же эффективную программу на Xilinx Vivado не удастся, хотя бы потому, что там нет конвейеров, управляемых программистом.

Заключение

Эта программа на языке HaSCoL была мною написана за 1 неделю, плюс несколько рабочих дней ушло на согласование интерфейсов, улучшение генератора кода на VHDL, исправление ошибок и т.п. Даже трудно себе представить, сколько времени понадобится инженеру, владеющему VHDL, на написание аналогичной программы с такими же глубокими конвейерами.

Несколько промышленных экспериментов мы уже провели. Например, программа вычисления сигнатур системы компьютерного стереозрения на языке HaSCoL была написана так же за 1 неделю, а инженер, работающий на полную ставку, писал на VHDL ту же программу больше полугода. На HaSCoL уже реализованы нейрокомпьютер, содержащий 500 нейронов в одном кристалле VIRTEX IV, процессор Macroblaze и еще несколько других больших систем. Это позволяет нам сделать вывод о действительно больших возможностях технологии.

```
if pk >= m1 && pl >=n1
then
ii=0;

  while ii < m
  do
1   inform (sum1(ii,pl)) | ii:=ii + 1
  done;
  skip;

  x:=ext(zz - (((zv >* zv) >> sh){0:29} :uint(30)), 32) | y:=0;
  j:= (30 :uint(5));
  while j >= 0
  do
    zzz:=((ext(y,32)<<2)+1)<<j | y:=y << 1;
2   if x >=zzz
    then y:=y + 1 | x:=x - zzz
    fi;
    j:=j-2
  done;
```

```
if y == 0 then y:=1 fi;

3 { if calca == 1
  { then
  {  az:=y | inform ina()

else
 kk = pk-m1 | ll = pl-n1 | ii:=0 |
 iii = if i+1 == m then 0 else i+1 fi;

4 { while ii < m
  { do
  {  inform(sum2(ext(ii, 10), iii, y, flagend, kk, ll)) | ii:=ii + 1
  { done

 fi
fi
}

5 { sum1(i,l)
  { {

sil = s[i][l] | s2il = s2[i][l];
zv:=if i == 0 then 0 else zv fi + ext(sil,22) |
zz:=if i == 0 then 0 else zz fi + ext(s2il,30)
}

  { sum2(ii,iii, y, flagend, kk, ll)
  { {
  { #define mj 0
  { #while mj < n
  {   aij(mj) = a[mj][ii] |
  {  let imj = iii + mj in
  {  { zij(mj) = z[if imj >= m then imj - m else imj fi][ii] } |
  { #if mj == 0
  {   F2 = (ext(az >* y, 44) :int(44)) |
  { #endif
  { #set mj $eval(mj+1)
  { #endw
  { skip;
  { #set mj 0
  { #while mj < n
6 {   FR(0,mj) = aij(mj) >* ((((ext(zij(mj), 22) << sh) :uint(22))-
```

```
6 { zv):
  int(22)) / F2 |
  #set mj $eval(mj+1)
  #endw
  #undef mj
  skip;
  #define halfm m
  #define mi 0
  #while mi < 6
  #set halfm $eval(halfm/2)
  #define mj 0
  #while mj < halfm
  #define mmj $eval(mj+mj)
    FR($eval(mi+1),mj) = FR(mi,mmj) >+ FR(mi,$eval(mmj+1)) |
  #set mj $eval(mj+1)
  #endw
    skip;
  #undef mj
  #set mi $eval(mi+1)
  #endw
  #undef mi
   F:= sext(FR(6,0) >+ FR(6,1), 52) + if ii == 0 then 0 else F fi;
   if F > maxF
   then
    maxF:=F | maxk:=kk | maxl:=ll
   fi;
   if flagend then inform(ans(maxk, maxl)) fi
  }
  end;
```

Список литературы

1. «Использование стандартного расширяемого алгоритмического языка в микропрограммировании», Н.Ф.Фоминых, сборник «Диалоговые микрокомпьютерные системы». М., МГУ, 1986.
2. «Архитектура среды визуального моделирования QReal», Системное программирование, вып. 4, 2009, Терехов А.Н., Брыксин Т.А., Литвинов Ю.В., Смирнов К.К., Никандров Г.А., Иванов В.Ю., Такун Е.И., кол-во страниц 26.
3. Фирменная документация: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_2/ug902-vivado-high-level-synthesis.pdf
4. «Эволюция наших взглядов на CoDesign за последние 15 лет», личный сайт: http://www.math.spbu.ru/user/ant/History_Evol_CODESIGN.pdf
5. «Using Hardware-Software Codesign Language to implement CANSCID», Oleg Medvedev, Ilya Posov, Formal Methods and Models for Codesign (MEMOCODE), 2010

8th IEEE/ACM International Conference on, pp 85 -88, http://oops.math.spbu.ru/dours/HaSCoL/MedvedevPosovMEMOCODE2010.pdf

6. «Accelerating multiple alignment on FPGA with a high-level hardware description language», Oleg Medvedev, Central Eastern European Software Engineering Conference in Russia, 2011, http://oops.math.spbu.ru/dours/HaSCoL/SECR11.pdf
7. «Обзор высокоуровневого языка разработки аппаратуры HaSCoL на примере клона процессора Xilinx Microblaze», Олег Медведев, Вторая научно-техническая конференция молодых специалистов «старт в будущее», 2011, с. 231–234, http://oops.math.spbu.ru/dours/HaSCoL/KBSM.pdf
8. «Hardware Description Language Based on Message Passing and Implicit Pipelining», Dmitri Boulytchev, Oleg Medvedev, EWDT 2009, http://oops.math.spbu.ru/dours/HaSCoL/EWDT09.pdf
9. «Представление связи исходного и выходного текстов при автоматическом порождении текстов программ», А.Е. Москаль, сборник «Автоматизированный реинжиниринг программ», СПб, Издательство Санкт-Петербургского университета, 2000 г.

О ЗАДАЧЕ ОПТИМАЛЬНОГО ВЫВЕДЕНИЯ РАКЕТЫ-НОСИТЕЛЯ НА ОКОЛОЗЕМНУЮ ОРБИТУ*

Думшева Т. Д., Кандоба И. Н., Костоусова Е. К., Ложников А. Б.

Институт математики и механики УрО РАН, Россия, Екатеринбург, kandoba@imm.uran.ru

Починский В. И.

ФГУП НПО автоматики им. академика Н. А. Семихатова, Россия, Екатеринбург

Аннотация. Доклад посвящен задачам управления, возникающим при выведении ракеты-носителя (носителя) на заданную околоземную эллиптическую орбиту. Движение носителя описывается нелинейной динамической системой. В ряде практически важных задач к искомому управлению предъявляются дополнительные требования. Первое заключается в максимизации выводимой на орбиту массы полезной нагрузки, второе диктуется необходимостью обеспечения успешного возвращения пилотируемого модуля носителя на Землю в случае аварийного прекращения его полета в любой момент времени. Второе условие приводит к возникновению ограничений на текущее фазовое состояние динамической системы.

Ключевые слова: ракета-носитель, управляемое движение, полезная нагрузка, квантиль, многопроцессорные вычислительные системы.

1. Введение

Одной из важнейших составляющих технологии подготовки ракеты-носителя (РН) к пуску является разработка и тестирование программного управления, обеспечивающего вывод РН на заданную орбиту. В настоящее время у разработчиков ракетно-космической техники наибольший интерес вызывают проблемы разработки такого управления, которое удовлетворяет следующим условиям: первое заключается в максимизации выводимой на орбиту массы полезной нагрузки, второе диктуется необходимостью в любой момент времени обеспечения успешного возвращения космического аппарата (КА) на Землю в случае аварийного прекращения полета РН. В частности, второе условие приобретает особую актуальность при под-

* Работа выполнена в рамках программ фундаментальных исследований Президиума РАН «Динамические системы и теория управления» и «Информационные, управляющие и интеллектуальные технологии и системы» при финансовой поддержке УрО РАН (12-П-1-1022, 12-П-1-1023), а также при поддержке программы ориентированных фундаментальных исследований УрО РАН (12-1-006-НПО) и интеграционного проекта УрО и СО РАН (12-С-1-1017).

готовке РН к пилотируемому пуску. В математической модели управляемого движения РН это условие приводит к возникновению ограничений на текущее фазовое состояние динамической системы. В том числе, задача построения такого оптимального управления активно исследуется на базе нелинейной математической модели управляемого движения РН [1]:

$$\begin{aligned}
&\dot{x}=v;\\
&\dot{v}=W(t,x,v,\vartheta,\psi);\\
&\dot{m}=-\nu(t); \qquad\qquad t\in[t_0,\, t_f] \quad t_0>0.\\
&\dot{\vartheta}=u_1(t);\\
&\dot{\psi}=u_2(t);
\end{aligned} \tag{1.1}$$

Здесь $x, v \in \mathbb{R}^3$ — координаты и скорости центра масс РН в некоторой инерциальной прямоугольной системе координат; m — масса РН; ν — заданная положительная кусочно-постоянная функция; ϑ, ψ — углы тангажа и рыскания РН; t_0 — момент начала движения РН; t_f — момент выхода РН на заданную орбиту; $W(t,x,v,\vartheta,\psi)=W_R(t,x,\vartheta,\psi)+W_A(t,x,v,\vartheta,\psi)+g(x)$ — ускорение (задается суммой составляющих, определяемых реактивными (W_R), аэродинамическими (W_A) и гравитационными (g) силами). В частности,

$$W_R(t,x,\vartheta,\psi)=\frac{P(t,x)}{m(t)}\,(\cos\vartheta\cos\psi,\ \sin\vartheta\cos\psi,-\sin\psi)^T,$$

где $P(t,x)$ — известная функция силы тяги двигателя РН.

В качестве управления в модели (1.1) используются угловые скорости $u_1(t)$ и $u_2(t)$ изменения углов тангажа ϑ и рыскания ψ соответственно — параметров, определяющих угловую ориентацию строительной оси РН. В этой модели учитываются как технические характеристики РН — параметры РН, так и свойства окружающей среды — параметры атмосферы. К параметрам РН относятся: массы основных ее конструктивных блоков (сухие массы ступеней); массы головного обтекателя и хвостового отсека; массы запасов топлива на каждой ступени; масса полезной нагрузки; скорости изменения массы топлива (расходы топлива) и величины удельных тяг двигателей на каждой ступени. Параметрами атмосферы являются ее основные термодинамические характеристики: температура, плотность, давление и скорость систематического ветра.

Орбита, на которую выводится РН, задается следующими пятью параметрами [1]: наклонение плоскости орбиты i; долгота восходящего узла Ω; минимальная высота орбиты $h_{\min}$; максимальная высота орбиты $h_{\max}$; аргумент перигея ω.

2. Вывод максимальной массы полезной нагрузки на заданную орбиту в детерминированных условиях

В [2] показано, что задача максимизации выводимой ракетой-носителем на заданную орбиту массы M полезной нагрузки равносильна задаче минимизации значения момента времени t_f выхода РН на заданную орбиту с фазовыми ограничениями в момент времени t_f — задаче оптимального быстродействия с терминальными ограничениями:

$$\min\{J[u(\cdot)] \mid u \in U\} \tag{2.1}$$

при ограничениях

$$|O(t_f) - \overline{O}| \leqslant \overline{\Delta}_O, \tag{2.2}$$

где

$$J[u(\cdot)] = t_f; \quad U = \{u \in \mathbb{R}^2 \mid |u_1| \leqslant u_1^{\max}, \ |u_2| \leqslant u_2^{\max}\};$$

$$u = (u_1, u_2)^\top \in \mathbb{R}^2, \quad u_1 = \dot{\vartheta}, \quad u_2 = \dot{\psi};$$

$$O(t_f) = (i(t_f), \Omega(t_f), h_{\max}(t_f), h_{\min}(t_f), \omega(t_f))^\top \in \mathbb{R}^5;$$

$$\overline{O} = (\overline{i}, \overline{\Omega}, \overline{h}_{\max}, \overline{h}_{\min}, \overline{\omega})^\top, \quad \overline{\Delta}_O = (\overline{\Delta}_i, \overline{\Delta}_\Omega, \overline{\Delta}_{h_{\max}}, \overline{\Delta}_{h_{\min}}, \overline{\Delta}_\omega)^\top \in \mathbb{R}^5,$$

где $\overline{i}$, $\overline{\Omega}$, $\overline{h}_{\max}$, $\overline{h}_{\min}$, $\overline{\omega}$ — желаемые значения параметров орбиты; $\Delta_i = i - \overline{i}$, $\Delta_\Omega = \Omega - \overline{\Omega}$, $\Delta_{h_{\max}} = h_{\max} - \overline{h}_{\max}$, $\Delta_{h_{\min}} = h_{\min} - \overline{h}_{\min}$, $\Delta_\omega = \omega - \overline{\omega}$ — отклонения параметров орбиты от желаемых значений; $\overline{\Delta}_i$, $\overline{\Delta}_\Omega$, $\overline{\Delta}_{h_{\max}}$, $\overline{\Delta}_{h_{\min}}$, $\overline{\Delta}_\omega$ — допустимые отклонения параметров орбиты от желаемых значений. Здесь векторные неравенства понимаются покомпонентно.

В работах [2–4] задача (2.1), (2.2) исследовалась в *детерминированных условиях* — при фиксированных значениях параметров РН и атмосферы в математической модели (1.1) управляемого движения РН.

В [2, 3] для задачи (2.1), (2.2) разработаны подходы к построению допустимого управления, обеспечивающего выведение РН на заданную орбиту к фиксированному моменту времени t_f. Допустимое управление в задаче (2.1), (2.2) строится как решение некоторой нелинейной экстремальной задачи. Для решения такой задачи предлагается итерационная процедура, основанная на методе сопряженных градиентов. Результаты вычислительного эксперимента показывают, что численная реализация этого подхода возможна лишь на высокопроизводительной вычислительной технике, обеспечивающей получение допустимого в задаче (2.1), (2.2) управления с удовлетворительной точностью за приемлемое время.

В [4] предложен один конструктивный метод численного решения задачи (2.1), (2.2) на безатмосферном участке полета РН. Основная идея этого метода заключается в декомпозиции движения РН на этом участке на три составляющих — боковое, вертикальное и горизонтальное движения РН. Для каждого из этих типов движения РН решается соответствующая специальная задача оптимального управления. Синтез решений этих задач позволяет построить оптимальное по быстродействию управление, обеспечивающее вывод РН на заданную орбиту. Описанный в [4] во многом базируется на инженерном представлении о структуре оптимального в задаче (2.1), (2.2) управления. Тем не менее, несмотря на отсутствие полного строгого с математической точки зрения обоснования этого алгоритма, результаты численного моделирования с использованием реальных подтверждают его эффективность.

3. Вывод максимальной массы полезной нагрузки на заданную орбиту в условиях случайных возмущений параметров

Сложившаяся практика такова, что работы по построению оптимального управления, обеспечивающего вывод РН на заданную орбиту, выполняются достаточно заблаговременно с использованием всех имеющихся данных об условиях планируемого пуска — параметров орбиты выведения, термодинамических характеристик атмосферы в районе космодрома, технических параметров РН и др. При этом это управление разрабатывается с учетом всех прогнозируемых условий, которые могут реализоваться в момент пуска, что приводит к необходимости учитывать в используемых математических моделях, во-первых, случайную природу свойств окружающей среды и, во-вторых, вероятные случайные отклонения значений параметров РН от проектируемых.

В [5] задача вывода на орбиту максимальной полезной нагрузки исследовалась в условиях *случайных возмущений параметров* модели (1.1) — рассматривалась ситуация, когда параметры РН и атмосферы математически определяются как *нормально распределенные случайные величины* с известными средними $E(\Pi)$ и среднеквадратическими отклонениями (СКО)$\sigma(\Pi)$ (Π — соответствующий параметр РН или атмосферы). При этом учитывается, что значения термодинамических характеристик атмосферы зависят от высоты H, широты φ и месяца года t^m: температура $T(H, \varphi, t^m)$; плотность $\rho(H, \varphi, t^m)$; давление $p(H, \varphi, t^m)$; две взаимно ортогональные составляющие скорости систематического ветра — зональная $V_{\mathrm{ZON}}(H, \varphi, t^m)$ и меридианальная $V_{\mathrm{MER}}(H, \varphi, t^m)$. Математические модели этих параметров атмосферы задаются общей формулой (см. [6])

$$A(H, \varphi, t^m) = E(A(H, \varphi, t^m)) + K_1(\varphi)K_2(t^m)(\alpha_1\chi_1(H) + \ldots + \alpha_l\chi_l(H)), \quad (3.1)$$

где $A(H, \varphi, t^m)$ — параметр атмосферы; $E(A(H, \varphi, t^m))$ — математическое ожидание параметра; α_k — некоррелированные нормально распределенные слу-

чайные величины с нулевым средним и единичной дисперсией; $\chi_k(H)$ — «координатные» функции аргумента H; l — количество координатных функций, задающих закон отклонения характеристики $A(H, \varphi, t^m)$ от $E(A(H, \varphi, t^m))$; $K_1(\varphi)$, $K_2(t^m)$ — заданные функции. При этом, для задания значений $T(H, \varphi, t^m)$, $\rho(H, \varphi, t^m)$, $p(H, \varphi, t^m)$ по формуле (3.1), используется один и тот же набор значений величин α_k ($k=1, 2, \ldots, l$), а для $V_{\text{ZON}}(H, \varphi, t^m)$ и $V_{\text{MER}}(H, \varphi, t^m)$ — разные наборы этих величин, отличные от вышеупомянутых.

Математические ожидания параметров РН и атмосферы, как и в [5], будем называть *номинальными значениями* соответствующих параметров или, короче, их *номиналами*, а априорно заданные значения номинала и СКО параметра РН — *проектными значениями* характеристик этого параметра.

В [5] предложена методика построения оценки максимальной массы полезной нагрузки, которая может быть выведена РН на заданную орбиту с вероятностью не меньше заданного порога, а также оценки величины выигрыша по массе выводимой на орбиту полезной нагрузки, который может быть получен за счет сужения неопределенности значений параметров РН и атмосферы непосредственно перед стартом. Эта методика базируется на использовании квантилей случайных величин, описывающих параметры модели (1.1). Для получения приближенных значений этих квантилей в [5] применяются известные [7] методы стохастического программирования, основанные на обработке и анализе статистических данных о реализовавшихся значениях соответствующих случайных величин. Наборы таких статистических данных формируются в результате проведения широкомасштабного вычислительного эксперимента на многопроцессорной вычислительной системе, в котором для различных значений параметров модели (1.1) численно моделируются пуски РН — управляемое движение РН от точки старта до момента выхода РН на заданную орбиту. В каждом пуске для определения оптимального по быстродействию управления РН используются описанные выше подходы, предложенные в [4].

В данной работе приводятся результаты исследований, начатых в [5]. Здесь для полноты изложения приводятся постановки исследуемых задач и краткое описание предложенных в [5] методов их решения. Также приводятся результаты такого численного эксперимента, полученные с использованием реальных данных.

З а д а ч а 1. Построение оценки максимальной массы полезной нагрузки, которая может быть выведена ракетой-носителем на заданную орбиту с вероятностью не меньше заданного порога P^*.

Содержательная постановка этой задачи может быть сформулирована следующим образом. На заданную орбиту в момент времени t_{f} выводится только последняя ступень РН, на которой сосредоточена полезная нагрузка и некоторые остатки топлива массы $M_T(t_{\text{f}})$ для двигателей этой ступени. За

счет уменьшения запасов топлива на последней ступени возможно увеличить массу выводимой на орбиту полезной нагрузки. Задача заключается в том, что для заданного порога вероятности P^* выхода РН на орбиту требуется определить верхнюю оценку дополнительной массы, на которую может быть увеличена масса выводимой на орбиту полезной нагрузки.

З а д а ч а 2. Определение величины выигрыша по массе выводимого на заданную орбиту полезного груза за счет уточнения параметров РН и идентификации параметров атмосферы перед стартом.

Под уточнением параметра РН понимается изменение проектных значений характеристик этого параметра — номинала и СКО.

Также непосредственно перед стартом могут быть идентифицированы параметры атмосферы — измерены значения ее термодинамических характеристик в районе космодрома. В результате этих измерений становятся известными законы изменения значений всех параметров атмосферы. Это знание математически выражается в том, что случайные величины α_k становятся точно известными, а, следовательно, «измеренные» функции $A(H, \varphi, t^m)$ — не случайными.

Выполнение перед стартом описанной процедуры потенциально позволяет получить выигрыш по массе выводимого на заданную орбиту груза. Под выигрышем понимается масса дополнительного груза, на которую может быть увеличена масса полезной нагрузки, выводимой на орбиту с вероятностью не ниже заданной.

3.1. Оценка максимальной массы полезной нагрузки, выводимой на орбиту с вероятностью не ниже заданной

Основная идея предложенного в [5] метода приближенного решения задачи 1 заключается в следующем. Выводимая на орбиту за время t_f масса РН $m_{ph} = m_{ph}(t_f)$ рассматривается как случайная величина, распределенная по неизвестному закону. Пусть $P(m) = P\{m_{ph} \leqslant m\}$ — функция распределения случайной величины m_{ph}, μ — сухая масса последней ступени, задаваемая с помощью нормально-распределенной случайной величины, M — фиксированная номинальная масса полезной нагрузки. В [5] показано, что задача определения максимальной полезной нагрузки при заданном уровне P^* вероятности вывода ее на орбиту сводится к вычислению λ^*-квантили $m_{\lambda^*} = \min\{m : P(m) \geqslant \lambda^*\}$ распределения $P(m)$ для $\lambda^* = 1 - P^*$. Требуемая оценка максимальной добавки ΔM^* к M будет равна:

$$\Delta M^* = m_{\lambda^*} - \mu_{P^*} - M. \tag{3.2}$$

Здесь μ_{P^*} — P^*-квантиль нормального распределения $N(E(\mu), \sigma^2(\mu))$ случайной величины μ, вычисляемый известным образом (см., например, [8]).

Задача об оценке λ^*-квантили $m_{\lambda^*}(\lambda^*=1-P^*)$ неизвестного распределения $P(m)$ в [5] решается с помощью статистической оценки $\widehat{m}_{\lambda^*}$ (см. [7]), полученной на основе объемного вычислительного эксперимента, описанного ниже.

3.2. Определение величины выигрыша по массе выводимого на заданную орбиту полезного груза

Приведенный в [5] приближенный метод определения величины выигрыша по массе выводимого на заданную орбиту груза за счет уточнения проектных значений характеристик параметра П, задаваемого в математической модели (1.1) управляемого движения РН нормально распределенной случайной величиной с математическим ожиданием $E(\Pi)$ и (СКО)$\sigma(\Pi)$, основан на представлении этого параметра в виде суммы двух независимых случайных величин

$$\Pi=\Pi^*+\delta\Pi. \tag{3.3}$$

Здесь Π^* — нормально распределенная случайная величина с математическим ожиданием $E(\Pi)$ и (СКО)$\sigma(\Pi^*)=\sqrt{1-\varepsilon^2}\sigma(\Pi^*)$, $\delta\Pi$ — нормально распределенная случайная величина с математическим ожиданием $E(\delta\Pi)=0$ и СКО $\sigma(\delta\Pi)=\varepsilon\sigma(\Pi)$, где $\varepsilon\in(0,1)$.

С помощью случайной величины Π^* моделируется процедура уточнения номинала параметра П, а с помощью случайной величины $\delta\Pi$ — процедура уточнения его СКО (изменения значения $\sigma(\Pi)$ умножением $\sigma(\Pi)$ на ε). При этом предполагается, что изменение (уточнение) проектных значений характеристик параметра П осуществляется независимо.

Суть предложенного в [5] приближенного метода определения выигрыша по массе выводимой на заданную орбиту полезной нагрузки за счет уточнения параметра П заключается в следующем. В математической модели (1.1) параметр П «разделяется» на два параметра Π^* и $\delta\Pi$, где значение случайной величины Π^* определяет уточненный номинал параметра П, а значение $\delta\Pi$ — отклонение значения параметра П от его уточненного номинала. Для параметра П при выбранном значении коэффициента ε строится разложение этого параметра по формуле (3.3). Соответствующий параметру П и значению коэффициента ε выигрыш по массе полезной нагрузки, выводимой на заданную орбиту с вероятностью не ниже P^*, предлагается в [5] вычислять по формуле

$$\delta M(\Pi,\varepsilon)=\Delta M^*(\Pi,\varepsilon)-\Delta M^*, \tag{3.4}$$

где $\Delta M^*(\Pi,\varepsilon)$ $(\varepsilon\in(0\ldots1))$ — величина, которая для параметра П в задаче 1 при фиксированной массе M полезной нагрузки и порога вероятности P^* ее успешного вывода на орбиту определяется равенством (3.2), а ΔM^* —

решение задачи 1 в ее исходной постановке, т. е. в постановке, в которой параметр П не «разделяется» на два параметра с помощью разложения по формуле (3.3).

Подобным образом (см. [5]) приближенно вычисляется и величина выигрыша по массе выводимой на орбиту полезной нагрузки за счет идентификации перед стартом параметров атмосферы в районе космодрома. Для этого фиксируются соответствующие различным термодинамическим характеристикам атмосферы [6] наборы A_0, A_{ZON}, A_{MER} значений случайных величин α_k ($k=1,2,\ldots,l$), использующихся в формуле (3.1) для задания соответствующих параметров атмосферы (A_0 — для $T(H, \varphi, t^m)$, $\rho(H, \varphi, t^m)$, $p(H, \varphi, t^m)$, A_{ZON} — для $V_{\text{ZON}}(H, \varphi, t^m)$, A_{MER} — для $V_{\text{MER}}(H, \varphi, t^m)$ (см. п. 3). Тогда соответствующий такой идентификации перед стартом параметров атмосферы выигрыш по массе полезной нагрузки, выводимой на заданную орбиту с вероятностью не ниже P^*, вычисляется по формуле

$$\delta M(A_0, A_{\text{ZON}}, A_{\text{MER}}) = \Delta M^*(A_0, A_{\text{ZON}}, A_{\text{MER}}) - \Delta M^*. \qquad (3.5)$$

Здесь $\Delta M^*(A_0, A_{\text{ZON}}, A_{\text{MER}})$ — величина, которая для наборов $\$A_0$, A_{ZON}, A_{MER} в задаче 1 определяется равенством (3.2), а ΔM^* — решение задачи 1 в постановке, где все значения случайных величин α_k ($k=1,2,\ldots,l$) задаются случайным образом с учетом законов их распределения.

3.3. Численное моделирование

Опишем результаты вычислительного эксперимента, в котором были реализованы описанные выше приближенные методы решения рассматриваемых в работе задач.

Исходные данные

В качестве исходных данных для проведения вычислительного эксперимента выступают термодинамические характеристики атмосферы, значения параметров РН, условия пуска и значение порога P^* вероятности успешного вывода РН на заданную орбиту.

Численно моделировались пуски РН типа «СОЮЗ-2» с космодрома Байконур и данных по атмосфере, соответствующих октябрю.

Приведем значения параметров орбиты выведения: наклонение плоскости орбиты ($i=90$ градусов); долгота восходящего узла ($\Omega=62{,}7318$ градусов); минимальная высота орбиты ($h_{\min}=220\,320$ метров); максимальная высота орбиты ($h_{\max}=229\,455$ метров); аргумент перигея орбиты ($\omega=62{,}0552$ градусов).

Порог P^* вероятности успешного вывода РН на заданную орбиту был принят равным 0,997.

В качестве значений номинальной массы полезной нагрузки и ее СКО были использованы статистические характеристики массы спутника «Corot» $E(M)=5\,881$ кг и $\sigma(M)=16.67$ кг. Этот спутник был выведен ракетой-носителем «СОЮЗ-2» с космодрома Байконур 27 декабря 2006 г. на околоземную стационарную эллиптическую орбиту, значения параметров которой приведены выше.

Аппаратное и программное обеспечение

Вычислительный эксперимент проводился на многопроцессорном вычислителе кластерного типа «Уран». Этот вычислитель собран на базе Blade серверов фирмы Hewlett-Packard. Он состоит из 208 вычислительных узлов, каждый из которых оснащен двумя четырехъядерными процессорами Intel Quad-Core Xeon, работающими на частоте 3.00 ГГц, и 16(32) гигабайтами оперативной памяти. В общей сложности пользователям доступно 1664 вычислительных ядра и 3584 Гбайт оперативной памяти. Система хранения вычислителя «Уран» позволяет разместить до 10 Тбайт данных. Для передачи данных между вычислительными узлами используется высокоскоростная сеть Infiniband с пропускной способностью 20 Гбит/сек.

Статистические данные

Для решения задач 1 и 2 был проведен статистический эксперимент с использованием метода Монте-Карло. Обработка результатов этого эксперимента позволила найти приближенные решения рассматриваемых задач. Для проведения эксперимента из значений параметров РН и термодинамических характеристик атмосферы формировался входной набор данных. При этом значения всех параметров генерировались случайным образом независимо друг от друга в соответствии с принятой для их описания вероятностной моделью.

Для приближенного решения задачи 1 был проведен вычислительный эксперимент, в котором было численно реализовано 236 000 модельных пусков РН. В каждом пуске строилось оптимальное по быстродействию управление, выводящее РН на заданную орбиту за минимальное время. В результате был сформирован массив статистических данных о реализовавшихся в пусках значениях массы РН m_{ph} и соответствующей ей массы остатков топлива $M_T(\overline{t}_{\text{f}})$ на третьей ступени в момент времени $\overline{t}_{\text{f}}$ выхода РН на орбиту. Здесь значение массы M полезной нагрузки было зафиксировано и во всех пусках РН равнялось номиналу (5 881 кг) этого параметра. А значения всех остальных параметров РН, задаваемых соответствующими нормально распределенными случайными величинами, и значения случайных величин α_k $(k=1,2,\ldots,l)$, которые используются в формуле (3.1) для задания соответствующих параметров атмосферы, генерировались случайным образом для каждого пуска. Таким образом был сформирован первый блок статистических данных.

Для определения приближенного значения выигрыша по массе выводимой полезной нагрузки, который может быть получен за счет идентифи-

кации параметров атмосферы непосредственно перед стартом, был сформирован второй блок статистических данных. Для получения этого блока данных было выполнено численное моделирование 356 000 пусков. Здесь в отличие от первого блока статистических данных значения всех термодинамических характеристик атмосферы задавались математическими ожиданиями $E(\alpha_k)=0$ $(k=1,2,\ldots,l)$ в соответствующих характеристикам наборах A_0, A_{ZON}, A_{MER} (см. п. 3.2). Таким образом, в задаче 2 моделировалась процедура идентификации перед стартом параметров атмосферы в районе космодрома.

Всюду нижний индекс в обозначениях величин указывает на соответствующий блок статистических данных.

3.4. Результаты численного моделирования

Приведем результаты обработки данных статистического эксперимента.

Оценка максимальной массы полезной нагрузки, выводимой на орбиту с вероятностью не ниже заданной

Для первого блока статистических данных на рис. 1 приведена гистограмма значений выводимой массы РН m_{ph}. Для этой случайной величины на рис. 1 приведены значения коэффициента асимметрии $(A)_1$ и величины эксцесса $(e)_1$. Они достаточно сильно отличаются от нуля. Это свидетельствует о том, что распределение значений случайной величины m_{ph} первого блока статистических данных далеко от нормального.

Описанная в п. 3.1 методика позволяет, используя формулу (3.2), вычислить приближенное значение величины $(\Delta M^*)_1$ максимального приращения

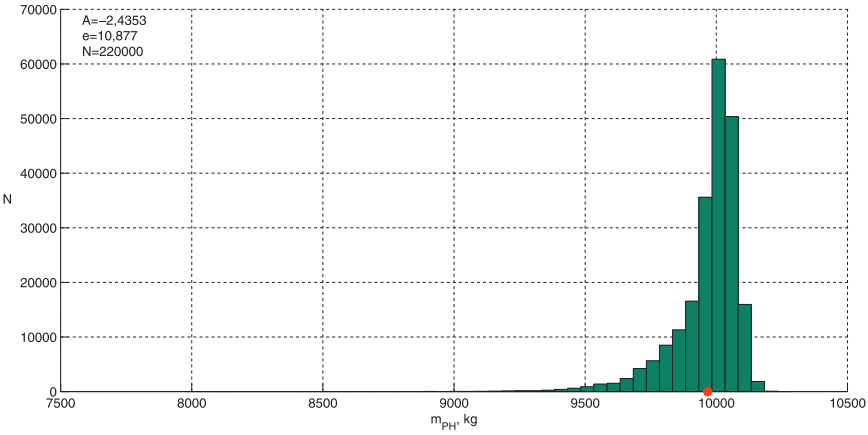


Рис. 1. Гистограмма значений m_{ph} выводимой массы РН в первом блоке статистических данных.

к массе $M = 5\,881$ кг полезной нагрузки, выводимой на заданную орбиту с вероятностью не ниже $P^* = 0{,}997$:

$$(\Delta M^*)_1 = 318{,}54 \text{ кг}.$$

Значение этой величины составляет около 5,5 % от указанного номинального значения выводимой на орбиту массы M полезной нагрузки.

Величина выигрыша по массе выводимого на заданную орбиту полезного груза за счет идентификации параметров атмосферы перед стартом

Для второго блока статистических данных на рис. 2 приведена гистограмма значений выводимой массы РН m_{ph}. На этом рисунке приведены значения коэффициента асимметрии $(A)_2$ и величины эксцесса $(e)_2$. Так же как и для первого блока статистических данных, распределение значений случайной величины m_{ph} второго блока достаточно далеко от нормального.

Применив описанную выше методику, нетрудно и для второго блока статистических данных определить приближенное значение максимального приращения $\Delta M^*(A_0, A_{\text{ZON}}, A_{\text{MER}})$ к массе M полезной нагрузки, выводимой на орбиту с вероятностью $P^* = 0{,}997$:

$$\Delta M^*(A_0, A_{\text{ZON}}, A_{\text{MER}}) = (\Delta M^*)_2 = 344{,}15 \text{ кг}.$$

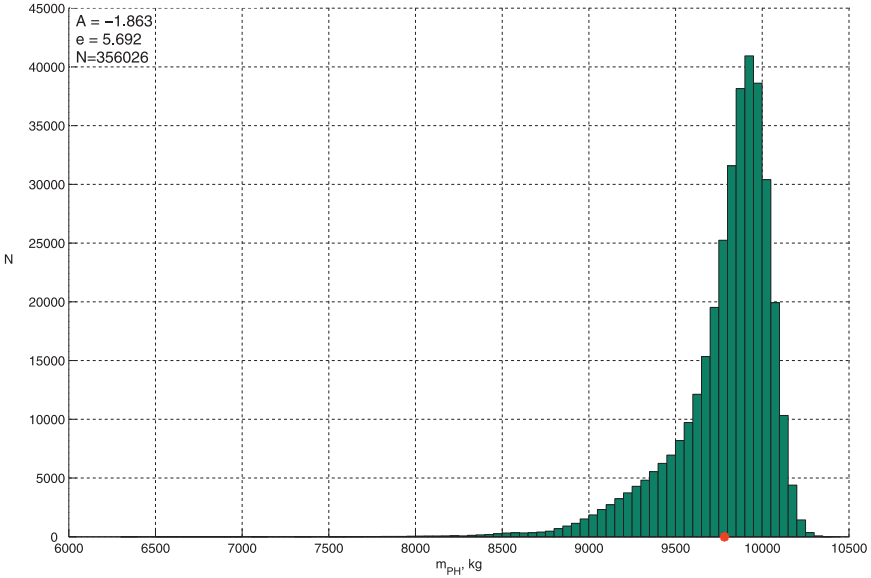


Рис. 2. Гистограмма значений m_{ph} выводимой массы РН во втором блоке статистических данных.

Тогда согласно (3.5) выигрыш по массе выводимой с вероятностью $P^* = 0{,}997$ на заданную орбиту полезной нагрузки, который может быть получен за счет идентификации перед стартом параметров атмосферы в районе космодрома, приближенно составит

$$\delta M(A_0, A_{\text{ZON}}, A_{\text{MER}}) = (\Delta M^*)_2 - (\Delta M^*)_1 = 25{,}61 \text{ кг.}$$

Эта величина составляет около 0,4 % от указанного номинального значения выводимой на орбиту массы M полезной нагрузки.

Что касается величины выигрыша по массе выводимой на заданную орбиту полезной нагрузки, который может быть получен за счет уточнения отдельного параметра РН, то анализ результатов проведенного статистического эксперимента показывает, что для большинства параметров РН изменение значений проектных характеристик соответствующего параметра не приводит к заметному выигрышу по массе выводимой полезной нагрузки.

4. Вывод полезной нагрузки на заданную орбиту при фазовых ограничениях

Рассмотрим задачу выведения на заданную эллиптическую орбиту РН с пилотируемым модулем. Предполагается, чтов случае возникновения непредвиденной аварийной ситуации пилотируемый модуль, иначе космический аппарат (КА), может отстыковаться от РН и совершить спуск на Землю. Неуправляемое движение КА от момента его отделения от РН до соприкосновения КА с земной поверхностью будем называть *аварийным спуском* КА. При движении КА в атмосфере возникает перегрузка, которая показывает во сколько раз аэродинамическое ускорение больше ускорения силы тяжести на поверхности Земли. Для перегрузки задается ее предельное значение, превышение которого может привести к нежелательным последствиям. Поэтому для обеспечения успешного возвращения КА выдвигаются дополнительные требования на перегрузки при аварийном спуске. Эти требования приводят к возникновению в задаче выведения РН на заданную орбиту фазовых ограничений на состояние управляемой динамической системы.

Математическая модель аварийного спуска КА описывает траекторию движения центра масс КА в неоднородном центральном поле тяготения. Параметрами модели являются масса, коэффициент лобового сопротивления и площадь миделя КА, а также — начальные условия в момент отделения КА от РН: пространственные координаты центра масс КА, направление и величина вектора скорости КА. Считается, что КА не обладает аэродинамическим качеством. Кроме того, предполагается, что начальные условия задаются значениями соответствующих фазовых координат РН в момент отделения КА.

4.1. Модель движения космического аппарата при аварийном баллистическом спуске

Приведем модель движения КА при аварийном спуске. Для того, чтобы различать соотношения, описывающие движение РН на всей траектории и движение КА при аварийном спуске, условимся величины, относящиеся к КА, снабжать символом «крышка».

Например, вектор с координатами центра масс КА и вектор скорости КА в инерциальной стартовой системе координат при аварийном спуске будем обозначать соответственно через $\hat{x}\in\mathbb{R}^3$ и $\hat{v}\in\mathbb{R}^3$, его массу — через $\hat{m}$, высоту полета КА над поверхностью Земли — через $\hat{H}$ и т. п.

Уравнения движения центра масс КА в инерциальной стартовой системе координат на промежутке $[\hat{t}_f, \hat{t}_0]$, могут быть записаны в следующем виде:

$$\dot{\hat{x}}=\hat{v};$$
$$\dot{\hat{v}}=\hat{W}_A(t,\hat{x},\hat{v})+\hat{g}(\hat{x}). \tag{4.1}$$

Здесь начальный момент $\hat{t}_0\in[t_0,t_f]$; момент $\hat{t}_f=\hat{t}_f(\hat{t}_0)$ соответствует моменту падения КА на Землю ($\hat{H}(\hat{t}_f)=0$). Начальные условия для этой системы определяются значениями векторов x и v для РН в момент $\hat{t}_0$. При этом, очевидно, имеют место равенства

$$\hat{x}(\hat{t}_0)=x(\hat{t}_0), \tag{4.2}$$

$$\hat{v}(\hat{t}_0)=v(\hat{t}_0). \tag{4.3}$$

Заметим, что иметь смысл исследовать также и случай, когда полагается

$$\hat{v}(\hat{t}_0)=v(\hat{t}_0)+\zeta(\hat{t}_0), \tag{4.4}$$

где $\zeta(\hat{t}_0)\in\mathbb{R}^3$ — некоторая добавка к скорости, которой также можно управлять наряду с $u(\cdot)$.

Полагаем, что масса КА $\hat{m}$ постоянна на промежутке $[t_0,t_f]$, а аэродинамическое ускорение $\hat{W}_A$ из (4.1) вычисляется по формуле

$$\hat{W}_A=-\kappa\rho(\hat{H})\|\hat{V}\|\hat{V}, \tag{4.5}$$

где κ — баллистический параметр (известный постоянный коэффициент), $\rho(\hat{H})$ — плотность атмосферы на высоте $\hat{H}$, $\hat{H}=H(\hat{x})$ — высота КА, $\hat{V}$ — вектор скорости центра масс КА относительно набегающего потока.

Важной величиной, связанной с динамикой спуска КА в атмосфере, является перегрузка, которая, напомним, показывает, во сколько раз аэроди-

намическое ускорение больше ускорения силы тяжести g_0 на поверхности Земли, и вычисляется по формуле

$$n = n(\hat{x}, \hat{v}) = \frac{\|\hat{W}_{\mathrm{A}}\|}{g_0} = \frac{\kappa\rho(\hat{H})\|\hat{V}\|^2}{g_0}. \tag{4.6}$$

Для обеспечения успешного возвращения КА на Землю в случае возникновения непредвиденной аварийной ситуации требуется, чтобы максимально возможные перегрузки при аварийном спуске не превышали заданного допустимого значения $\overline{n}_{\max}$. При этом приходится рассматривать целый ансамбль возможных траекторий аварийного спуска, каждая из которых начинается в некоторый момент $\hat{t}_0 \in [t_0, t_{\mathrm{f}}]$ на траектории РН. Чтобы подчеркнуть зависимость траектории $\hat{x}(t)$, $\hat{v}(t)$ от момента $\hat{t}_0$ начала аварийного спуска, введем обозначения

$$\hat{x}(t) = \hat{x}(t; \hat{t}_0), \quad \hat{v}(t) = \hat{v}(t; \hat{t}_0); \tag{4.7}$$

тогда ограничение на перегрузки может быть записано в виде

$$\max_{\hat{t}_0 \in [t_0, t_{\mathrm{f}}]} n_{\max}(\hat{x}(\hat{t}_0), \hat{v}(\hat{t}_0)) \leqslant \overline{n}_{\max}, \tag{4.8}$$

где

$$n_{\max}(\hat{x}(\hat{t}_0), \hat{v}(\hat{t}_0)) = \max_{t \in [\hat{t}_0, \hat{t}_{\mathrm{f}}]} n(\hat{x}(t; \hat{t}_0), \hat{v}(t; \hat{t}_0)), \quad \hat{t}_0 \in [t_0, t_{\mathrm{f}}].$$

Далее функцию $n_{\max} = n_{\max}(\hat{x}(\hat{t}_0), \hat{v}(\hat{t}_0))$ будем называть *функцией максимальной перегрузки*.

4.2. Функция максимальной перегрузки на базовой траектории выведения РН

Было проведено численное исследование свойств функции максимальной перегрузки $n_{\max}$ на *базовой траектории выведения РН* — траектории РН, которая соответствует построенному на ФГУП НПО автоматики им. академика Н.А. Семихатова для приведенных в п. 3 исходных данных *базовому управлению*. Точки этой траектории во всех вычислительных экспериментах использовались как начальные точки аварийного спуска КА, в которых задавались различные начальные условия.

На рис. 3 приведена зависимость значений функции максимальной перегрузки $n_{\max}$ вдоль базовой траектории выведения РН от высоты $\hat{H}$, на рис. 4 — от времени $\hat{t}_0 \in [t_0, t_{\mathrm{f}}]$. На этих рисунках горизонтальной линией красного цвета отмечено пороговое значение перегрузки $n_{\max} = 9$.

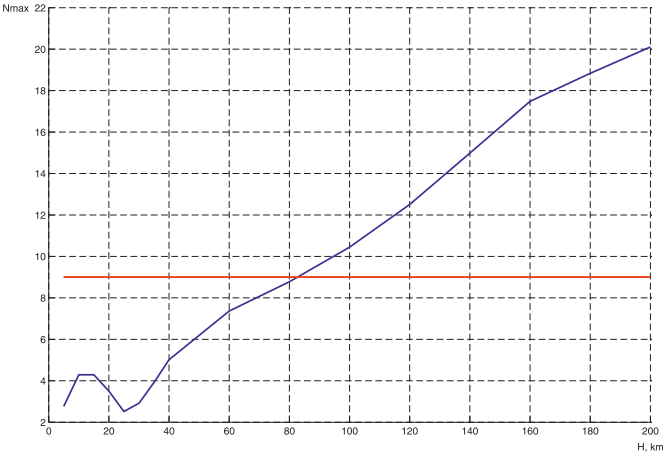


Рис. 3. Зависимость значений функции максимальной перегрузки $n_{\max}$ вдоль базовой траектории выведения РН от высоты $\widehat{H}$.

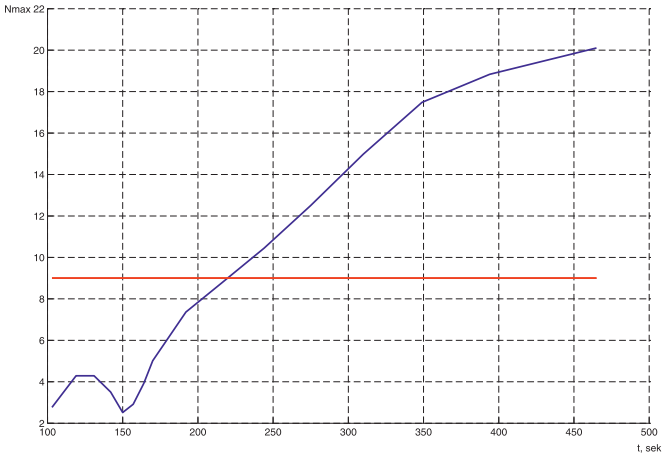


Рис. 4. Зависимость значений функции максимальной перегрузки $n_{\max}$ вдоль базовой траектории выведения РН от времени $\widehat{t}_0 \in [t_0, t_f]$.

Анализ этих данных позволяет сделать вывод, что для заданной пороговой величины $\overline{n}_{\max} = 9$ максимальной перегрузки $n_{\max}$ ограничения (4.8) выполняются до высоты $H = 82$ км полета РН — на всем участке работы первой ступени РН и части участка работы второй ступени. На другой части работы второй ступени РН (до высоты $H = 120$ км) наблюдается срав-

нительно небольшое превышение величины максимальной нагрузки ее пороговой величины — до 4 единиц. На остальной части базовой траектории выведения РН (большей части работы второй ступени РН и всем участке работы его третьей ступени) максимальная перегрузка превосходит $\overline{n}_{\max}=9$ на 6–10 единиц. Этот вывод привел к необходимости проведения наиболее детального численного исследования функции максимальной перегрузки $n_{\max}$ на той части базовой траектории выведения РН, которая соответствует высотам H превышающим 80 км.

Значения функции $n_{\max}$ определяются фиксированными параметрами системы (4.1), описывающей движение КА при аварийном спуске, и начальными условиями $\hat{x}(\hat{t}_0),\hat{v}(\hat{t}_0)$ для этой системы, т. е. можно считать, что $n_{\max}$ зависит от шести параметров — компонент векторов $\hat{x}(\hat{t}_0),\hat{v}(\hat{t}_0)$. Здесь для выделения тех ресурсов, используя которые возможно удовлетворить ограничениям (4.8) по максимальной перегрузке, целесообразно сначала определить основные более конструктивные факторы, оказывающие доминирующее влияние на значения функции максимальной перегрузки $n_{\max}$. Нетрудно доказать следующую теорему.

Т е о р е м а 1. Пусть движение КА осуществляется в центральном поле силы тяжести в предположении, что Земля — это шар радиуса R, без учета вращения Земли и наличия систематического ветра. Тогда

$$n_{\max}=n_{\max}(H, \mathrm{v}, \alpha), \tag{4.9}$$

где H — высота, на которой находится КА в момент начала аварийного спуска, $\mathrm{v}=\|\hat{v}(\hat{t}_0)\|$ — модуль скорости КА в этот момент и α — угол между $\hat{x}(\hat{t}_0)$ и $\hat{v}(\hat{t}_0)$.

Результаты численного моделирования для базовой траектории выведения РН показывают, что значения максимальных перегрузок при отсутствии ветра отличаются от соответствующих значений функции $n_{\max}$, вычисленных при наличии систематического ветра, в среднем не более, чем на 1,5 %. Полученные результаты позволяют сделать и вывод о том, что угловая скорость вращения Земли и отсутствие центральной симметрии у поля тяжести Земли оказывает существенно меньшее влияние на значения функции $n_{\max}$, чем параметры H, v, α. Это означает, что зависимость $n_{\max}=n_{\max}(H, \mathrm{v}, \alpha)$ является достаточно адекватной математической моделью для исследования свойств функции максимальной перегрузки.

4.2.1. Свойства функции максимальной перегрузки $n_{\max}(H, \mathrm{v}, \alpha)$

Исследование свойств функции максимальной перегрузки $n_{\max}$ выполнено с помощью крупномасштабного вычислительного эксперимента, в рам-

ках которого осуществлялось численное моделирование аварийных спусков КА с точек базовой траектории выведения РН. В этом исследовании функция $n_{\max}$ рассматривалась как функция трех аргументов: $n_{\max}=n_{\max}(H,\mathrm{v},\alpha)$. Значения аргументов этой функции варьировались в соответствующих диапазонах со следующими шагами:

1. $H\in S_H=[5{,}200]$ км: $\Delta H=5$ км ($5\leqslant H<40$ км), $\Delta H=20$ км ($40\leqslant H<200$ км);
2. $\mathrm{v}\in S_{\mathrm{v}}=[0{,}6000]$ м/с: $\Delta\mathrm{v}=500$ м/с;
3. $\alpha\in S_\alpha=[0{,}150]$ град: $\Delta\alpha=5$ град.

На базовой траектории выведения РН с указанным выше шагом ΔH по высоте ($5\leqslant H<200$ км) задавались начальные точки $\hat{x}(\hat{t}_0)=x(\hat{t}_0)$ аварийного спуска КА. Для каждой такой точки с помощью параметров v и α, значения которых варьировались в указанных выше соответствующих диапазонах с соответствующими шагами, задавались начальные условия (4.3) для скорости $\hat{v}(\hat{t}_0)$ (см. п. 4.1). При этом вектор $\hat{v}(\hat{t}_0)$ строился в плоскости, определяемой векторами $x(\hat{t}_0)$ и $v(\hat{t}_0)$ — положением и скоростью РН в момент начала аварийного спуска. В результате на $D_{H\mathrm{v}\alpha}$ была построена сеточная аппроксимация функции

$$n_{\max}(H,\mathrm{v},\alpha)=\max_{t\in[\hat{t}_0,\hat{t}_f]}\frac{\kappa\rho(t)\mathrm{v}^2(t)}{g_0},\qquad(4.10)$$

где $\rho(t)$, $\mathrm{v}(t)$ — плотность атмосферы и скорость КА в соответствующей моменту времени t точке траектории аварийного спуска КА. Выше через $D_{H\mathrm{v}\alpha}$ обозначен параллелепипед, полученный как прямое произведение отрезков S_H, S_{v}, S_α — $D_{H\mathrm{v}\alpha}=S_H\times S_{\mathrm{v}}\times S_\alpha$.

На рис. 5, 6 приведены в разных ракурсах фрагменты графиков функций $n_{\max}(H^\circ,\mathrm{v},\alpha)$ для $H^\circ=80$, 100, 120 км. На них красным цветом отображен график функции $n_{\max}(80,\mathrm{v},\alpha)$, зеленым цветом — $n_{\max}(100,\mathrm{v},\alpha)$, голубым цветом — $n_{\max}(120,\mathrm{v},\alpha)$.

Анализ данных, полученных в результате вычислительного эксперимента, позволяет сформулировать следующие свойства функции $n_{\max}(H,\mathrm{v},\alpha)$:

1. Функция $n_{\max}(H,\mathrm{v},\alpha)$ является монотонно-возрастающей по аргументам H и v:

$$\frac{\Delta n_{\max}}{\Delta\mathrm{v}}(H,\mathrm{v},\alpha)>0\ \ \forall\Delta\mathrm{v}>0,\quad \frac{\Delta n_{\max}}{\Delta H}(H,\mathrm{v},\alpha)>0\ \ \forall\Delta H>0;$$

2. Скорость возрастания функции $n_{\max}(H,\mathrm{v},\alpha)$ по аргументам H и v тем выше, чем больше величина $|\alpha-\alpha^*|$, где $\alpha^*=\pi/2$;

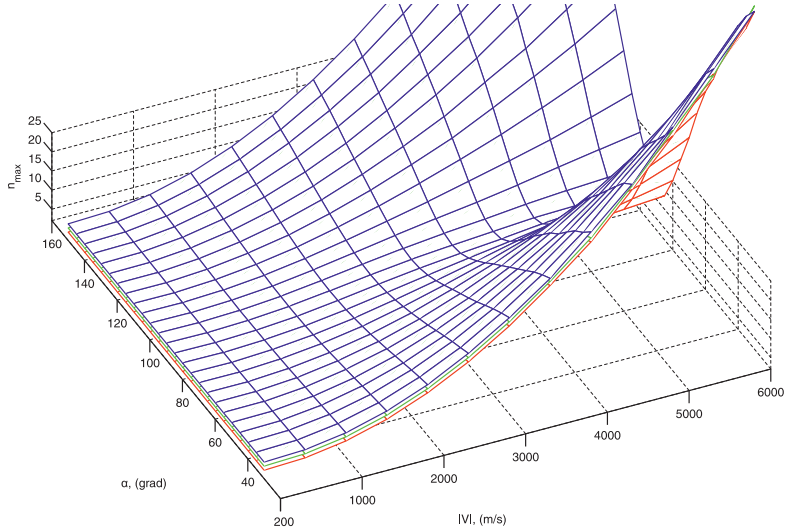


Рис. 5. Фрагменты графиков функций $n_{\max}(H^{\circ}, \mathrm{v}, \alpha)$ для $H^{\circ} = 80$; 100; 120 км (для $H^{\circ} = 80$ — красным цветом, для $H^{\circ} = 100$ — зеленым цветом, для $H^{\circ} = 120$ — голубым цветом)

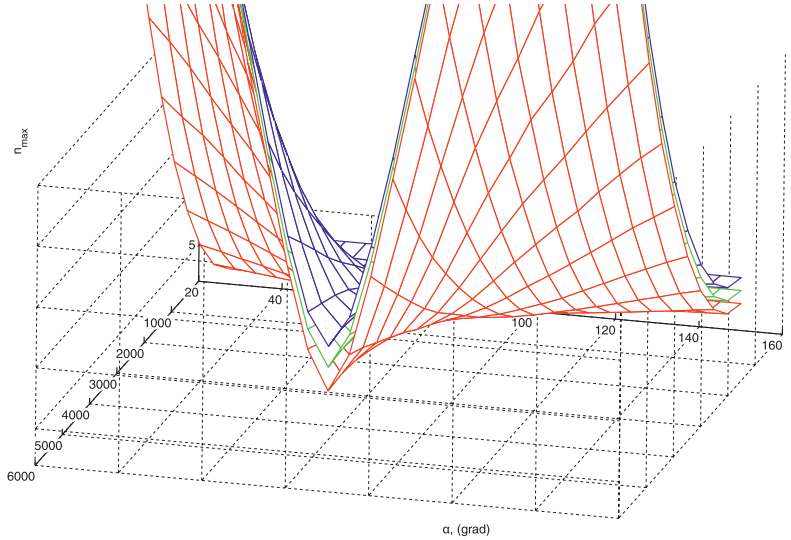


Рис. 6. Фрагменты графиков функций $n_{\max}(H^{\circ}, \mathrm{v}, \alpha)$ для $H^{\circ} = 80$, 100, 120 км (для $H^{\circ} = 80$ — красным цветом, для $H^{\circ} = 100$ — зеленым цветом, для $H^{\circ} = 120$ — голубым цветом)

3. Ограничения (4.8) по максимальной перегрузке выполняются до высоты $H^*=80$ км полета РН;
4. Точка $\alpha^*=\pi/2$ является точкой минимума для функции $n_{\max}(H^\circ, \mathrm{v}^\circ, \alpha)$ для всех $H^\circ \geqslant H^*$ и $\mathrm{v}^\circ>0$.

Поскольку для базовой траектории выведения РН ограничения (4.8) по максимальной перегрузке выполняются до высоты $H^*=80$ км полета РН, то дальнейшее детальное численное исследование свойств функции $n_{\max}$ проводилось на участке базовой траектории выведения РН, который соответствует высотам превышающим H^*.

На рис. 7 отображен фазовый портрет сужения $n_{\max}(80, \mathrm{v}, \alpha)$ функции $n_{\max}(H=80$ км). На этом графике символом «звезда» обозначена точка, соответствующая значению максимальной перегрузки, возникающей при аварийном спуске КА с точки базовой траектории выведения РН на высоте $H=80$ км с начальными условиями (4.2), (4.3) (см. п. 4.1) в этой точке. Аналогичные фазовые портреты функции $n_{\max}(H^\circ, \mathrm{v}, \alpha)$ для $H^\circ=100$ км, $H^\circ=120$ км и $H^\circ=140$ км приведены на рис. 8, 9 и 10 соответственно.

Перечисленные выше свойства функции $n_{\max}(H, \mathrm{v}, \alpha)$ позволяют сделать следующий важный вывод — аргументы v и α этой функции являются теми «рычагами», «манипулируя» которыми можно попытаться удовлетворить ограничениям (4.8) по максимальной перегрузке на высотах $H>H^*$ полета РН.

Во-первых. Нетрудно видеть (см. рис. 7–10), что для всех $H>H^*$ и $\mathrm{v}>0$ чем ближе значение α к α^* ($\alpha^*=\pi/2$), тем меньше значение максимальной перегрузки $n_{\max}$. Это свойство функции $n_{\max}(H, \mathrm{v}, \alpha)$ адекватно отражает физику процесса аварийного спуска КА. Поскольку α определяет угол, под которым при аварийном спуске КА входит в плотные слои атмосферы, то, чем ближе траектория движения КА к касательной к «границе» атмосферы, тем меньше «удар КА об атмосферу» и тем больше возможностей у КА для сброса скорости прежде чем траектория его движения приблизится к «наиболее крутому» ее участку, на котором и реализуется максимум перегрузки (см. рис. 11). А чем ближе угол α к α^*, тем более «пологo» при аварийном спуске КА входит в плотные слои атмосферы. Указанное свойство функции $n_{\max}(H, \mathrm{v}, \alpha)$ и приведенные рассуждения приводят к идее рассмотрения аргумента α функции $n_{\max}(H, \mathrm{v}, \alpha)$ как дополнительного ресурса, использование которого при построении допустимого управления может привести к удовлетворению ограничений (4.8) по перегрузке при аварийном спуске КА при аварийном спуске КА с любой точки траектории движения РН до максимально возможной высоты полета РН (момента времени $t^* \in [t_0, t_\mathrm{f}]$).

Во-вторых. Данные, отображенные на рис. 8–10 показывают, что на высотах $>H^*$ «мгновенное» уменьшение значения модуля скорости $\mathrm{v}=|\hat{v}(\hat{t}_0)|$ РН в момент времени $\hat{t}_0$ (*тормозной импульс* (4.4)) формально приводит к удовлетворению ограничений (4.8) по максимальной перегрузке при аварийном

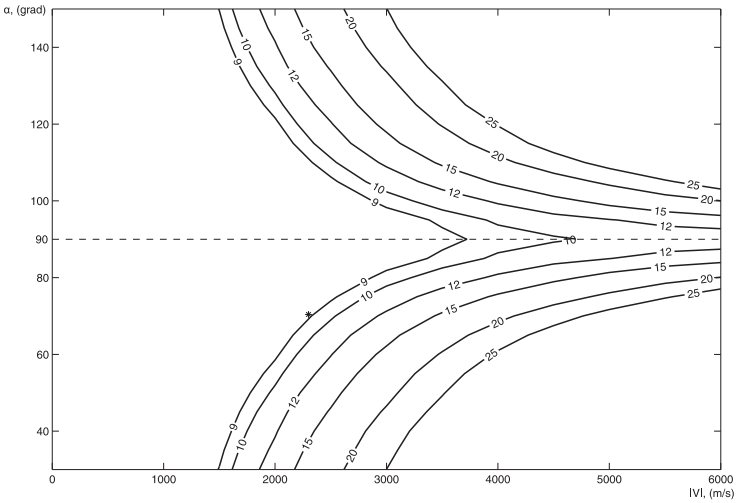


Рис. 7. Линии уровня функции $n_{\max}(H^{\circ}, \mathrm{v}, \alpha)$ (высота $H^{\circ} = 80$ км)

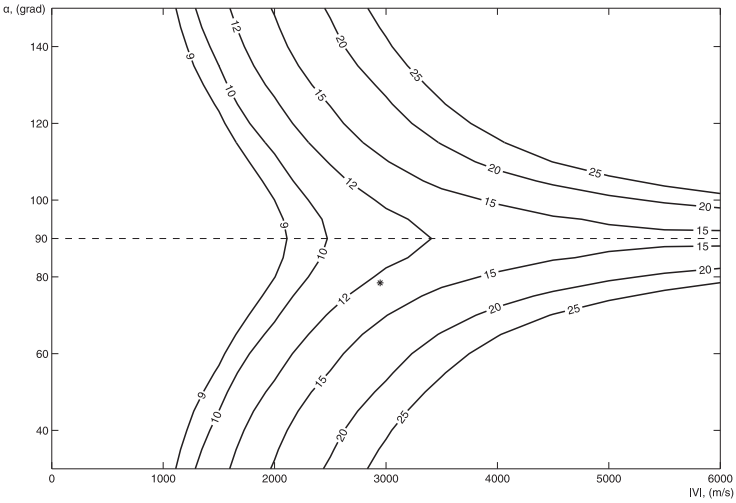


Рис. 8. Линии уровня функции $n_{\max}(H^{\circ}, \mathrm{v}, \alpha)$ (высота $H^{\circ} = 100$ км)

спуске КА с соответствующей $\widehat{t}_0$ точки на базовой траектории выведения РН. В частности, на высоте $H = 100$ км точки базовой траектории выведения РН такой тормозной импульс должен обеспечивать «сброс» величины скорости РН приблизительно со значения 2500 м/с до значения 2100 м/с за счет «мгновенного» уменьшения скорости на величину $|\Delta \mathrm{v}| \approx 400$ м/с (см. рис. 8). На вы-

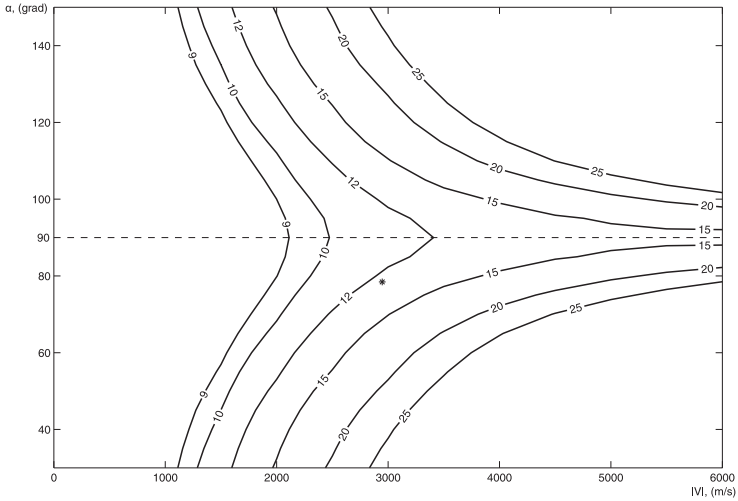


Рис. 9. Линии уровня функции $n_{\max}(H^{\circ}, \mathrm{v}, \alpha)$ (высота $H^{\circ} = 120$ км)

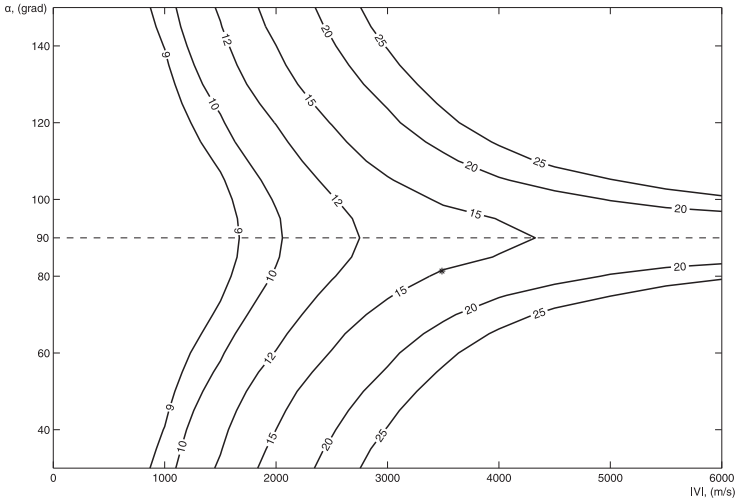


Рис. 10. Линии уровня функции $n_{\max}(H^{\circ}, \mathrm{v}, \alpha)$ (высота $H^{\circ} = 140$ км)

соте $H = 120$ $|\Delta \mathrm{v}| \approx 1\,000$ м/с (см. рис. 9), на высоте $H = 140$ $|\Delta \mathrm{v}| \approx 2\,000$ м/с (см. рис. 10). С ростом высоты H величина $|\Delta \mathrm{v}|$ достаточно быстро растет. Однако возникающая при этом проблема заключается в том, что такой тормозной импульс приводит к возникновению в момент времени t_0 перегрузки, значительно превышающей ее пороговую величину $n_{\max} = 9$. Следовательно,

в рассматриваемой ситуации, тормозной импульс не способен привести к желаемому результату. Тем не менее, теоретически возможно, что аргумент v функции $n_{\max}(H, \mathrm{v}, \alpha)$ может быть использован как дополнительный ресурс (см. (4.4)) при построении допустимого управления.

4.3. Подходы к нахождению допустимых управлений в задаче вывода РН на заданную орбиту при наличии ограничений на перегрузки

Результаты численного моделирования свидетельствуют о том, что не исключена возможность, что в задаче вывода РН на заданную орбиту не существует допустимое управление, удовлетворяющее ограничениям (4.8) по перегрузкам на всем интервале $[t_0, t_f]$ — величины v объективно велики на больших высотах при полете РН с постоянно работающим двигателем. Поэтому, принимая модель (4.1)–(4.3) для аварийного спуска КА, можно попытаться за счет выбора управления $u(\cdot)$ приблизить значения α к $\alpha^* = \pi/2$ на «средних высотах», чтобы отодвинуть вправо (отдалить) момент, когда величины перегрузок становятся недопустимыми.

Следовательно, практический интерес представляет следующая задача:

З а д а ч а 3. Для управляемой системы (1.1) с заданными начальными значениями найти допустимое программное управление $u(\cdot) \in U$ (см. п. 2) на некотором промежутке времени $[t_0, t_f]$ такое, что выполнены условия (2.2) на параметры орбиты и ограничения (4.8) на перегрузки на некотором промежутке времени $[t_0, t^*]$, $t_0 < t^* \leqslant t_f$.

Заметим, что

$$\cos\alpha = \frac{\hat{x}(\hat{t}_0)^\top \hat{v}(\hat{t}_0)}{\|\hat{x}(\hat{t}_0)\| \, \|\hat{v}(\hat{t}_0)\|}. \tag{4.11}$$

Поэтому можно ввести функцию

$$\chi(x, v) = \frac{(x^\top v)^2}{\|x\|^2 \, \|v\|^2} = \chi(x). \tag{4.12}$$

и для нахождения допустимых управлений на фиксированном промежутке $[t_0, t_f]$ минимизировать функционал типа

$$J[u(\cdot)] = \Phi(x(t_f)) + \beta \int_{t_*}^{t^*} \gamma(t) \chi(x(t)) dt. \tag{4.13}$$

Здесь $\Phi(x(t_f))$ — функционал, минимизация которого обеспечивает выполнение ограничений (2.2) на параметры орбиты выведения; $\gamma(t)$, $t \in [t_0, t_f]$ — выбранная неотрицательная весовая функция, отличная от 0 только на

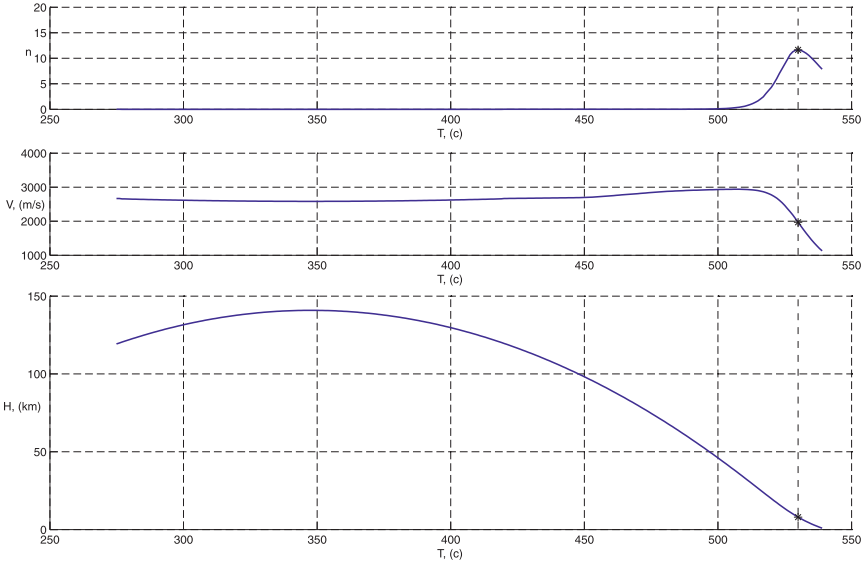


Рис. 11. Зависимость перегрузки n, скорости v, и высоты H от времени $t \in [\hat{t}_0, \hat{t}_f]$ на траектории аварийного спуска КА с точки на высоте $H(\hat{t}_0) = 120$ км базовой траектории выведения РН. На графиках «звездой» указаны точки соответствующие максимальному значению перегрузки

некотором промежутке $[t_*, t^*] \subset [t_0, t_f]$. Числа t_*, t^* можно задать, пользуясь графиком зависимости $n_{\max}(H)$ для траектории, соответствующей базовому управлению (см. рис. 3). А именно, целесообразно взять t_* равным значению, соответствующему $H \approx 80$ км (где величины перегрузок еще допустимы), а t^* — соответствующим, например, $H \approx 120$ км.

Здесь коэффициент $\beta > 0$ выполняет роль штрафного коэффициента. Таким образом, следуя общей идее метода внешних штрафов [9, 10], задачу минимизации функционала (4.13) нужно решать для некоторой возрастающей последовательности $\beta = \beta_0$, $\beta = \beta_1 > \beta_0$, …, а в качестве начального приближения к управлению на очередном шаге брать управление, полученное на предыдущем шаге. Значение β_0 можно взять равным нулю.

Для численной минимизации функционала (4.13) можно, например, воспользоваться методом сопряженных градиентов. Здесь следует заметить, что минимизация функционала (4.13) с интегральным слагаемым, строго говоря, не обязана обеспечивать поточечное выполнение ограничений (4.8). Тем не менее можно использовать этот метод для нахождения решения задачи 3 хотя бы при не очень больших значениях $t^* - t_0$.

Заключение

Результаты численного моделирования с использованием реальных данных позволяют сделать следующие выводы:

1. Учет статистических характеристик случайных величин, моделирующих случайные возмущения параметров математической модели управляемого движения РН, позволяет получить значимую оценку для максимального приращения к массе полезной нагрузки, выводимой ракетой-носителем на орбиту с вероятностью не ниже заданной.

2. Измерение непосредственно перед стартом значений термодинамических характеристик атмосферы в районе космодрома способно привести к заметному выигрышу по массе выводимой полезной нагрузки. Уточнение отдельно взятого параметра РН для большинства таких параметров не приводит к существенному выигрышу по массе выводимой полезной нагрузки. Представляется целесообразным исследовать вопрос о величине такого выигрыша в случае совместного уточнения группы параметров РН, что является одним из направлений будущих исследований.

3. Рассмотрены подходы к численному решению задачи выведения РН на заданную орбиту с учетом фазовых ограничений, основанные на решении вспомогательных задач оптимального управления на фиксированном промежутке времени и использовании идеи метода штрафов. При этом минимизируемый функционал во вспомогательной задаче отвечает за выполнение ограничений на параметры орбиты, а фазовые ограничения, связанные с перегрузками, учитываются путем введения тех или иных слагаемых со штрафами. На основе анализа результатов проведенного вычислительного эксперимента предлагается для решения задачи 3 произвести неявный учет фазовых ограничений путем введения в минимизируемый функционал слагаемого, зависящего от угла α между радиусом-векторам x и скоростью v.

Литература

1. *Охоцимский Д.Е, Сихарулидзе Ю.Г.* Основы механики космического полета. М.: Наука, 1990.
2. *Думшева Т.Д., Костоусов В.Б., Костоусова Е.К., Починский В.И.* Исследование задачи оптимального вывода полезной нагрузки на заданную эллиптическую орбиту// Тр. Ин-та математики и механики УрО РАН. 2010. Т.16, № 5. С. 57–65.
3. *Костоусова Е.К., Починский В.И.* О задачах выведения ракеты-носителя на заданные эллиптические орбиты // Тр. Ин-та математики и механики УрО РАН. 2011. Т.17, № 3. С. 201–216.
4. *Мазгалин Д.В.* Построение способа управления ракетой-носителем при использовании в качестве управления программных угловых скоростей разворотов // Информационно-управляющие системы. 2010. № 3(46). С. 21–29.
5. *Кандоба И.Н., Козьмин И.В., Костоусов В.Б., Ложников А.Б., Починский В.И.* О задаче вывода на орбиту максимальной полезной нагрузки в условиях случайных возмущений параметров //Автоматика и телемеханика. 2012. № 3. С. 52–63.

6. Ракеты и ракеты-носители // Отраслевой стандарт. Методика задания горизонтальной скорости ветра и термодинамических параметров атмосферы в районе полигона «Байконур» в диапазоне высот 1–120 км. ОСТ 92-5165-92.
7. *Кибзун А.И., Кан Ю.С.* Задачи стохастического программирования с вероятностными критериями. М.: Физматлит, 2009.
8. *Гмурман В.Е.* Теория вероятностей и математическая статистика. М.: Высш. шк., 2003.
9. *Мину М.* Математическое программирование. Теория и алгоритмы. М.: Наука, 1990.
10. *Васильев Ф.П.* Численные методы решения экстремальных задач. М.: Наука, 1980.

ТЕХНОЛОГИЯ ГИДОВ ДЛЯ РЕШЕНИЯ ПРОБЛЕМЫ ВЗРЫВА СОСТОЯНИЙ ПРИ ПРОЕКТИРОВАНИИ ПРОМЫШЛЕННЫХ СИСТЕМ

Котляров В.П., Дробинцев П.Д.

Санкт-Петербургский государственный политехнический университет, e-mail: vpk@spbstu.ru

Аннотация. В работе рассматривается технология гидов для решения проблем взрыва состояний при проектировании промышленных систем. Гид — абстрактный сценарий в модели проектируемой системы, направляющий генерацию детальных поведенческих трасс. Основная задача гидов — существенно сократить пространство состояний, используемое верификатором в процессе генерации трасс. В статье описан методы отслеживания покрытия требований, проверки сохранения семантики требований, метод поиска причин ошибок при генерации трасс. Приведена оценка результатов использования технологии гидов при проектировании промышленных проектов.

Ключевые слова: проектирование спецификаций, взрыв состояний, требования, поведенческая модель, трасса, гид.

Введение

Среди основных проблем автоматизации разработки и тестирования программного обеспечения промышленных приложений в виде сосредоточенных модулей или распределенных сетей непременно отмечается проблема обработки сложных и больших по объему спецификаций требований. Документы, фиксирующие спецификации требований пишутся, как правило, на естественном языке и могут содержать сотни и тысячи пунктов требований. В силу этого задача формализации требований для описания поведенческих сценариев, используемых для разработки автоматических тестов или ручных тестовых процедур, характеризуется как задача огромной сложности и трудоемкости.

Применимость формальных методов в промышленности в огромной степени определяется тем, насколько адекватен язык формализации принятой инженерной практике, в которую вовлечены не только собственно разработчики кода и тестировщики, но и заказчики, руководители проектов разных уровней, маркетологи и другие специалисты. Ясно, что никакой чисто логический язык не подходит для адекватной формализации требований, которая бы одновременно сохранила семантику разрабатываемого приложения и удовлетворяла всех «причастных лиц» [1].

В современной проектной документации формулировка исходных требований задается либо конструктивно, когда из текста требования на естественном языке удается реконструировать процедуру контроля или сце-

нарий проверки выполнения данного требования, либо неконструктивно, когда заданное в требовании свойство, не содержит пояснения способа его проверки.

Например, поведенческие требования телекоммуникационных приложений, при наличии описанного сценария выполнения, являются конструктивно заданными и допускают для проверки реализуемости использование верификации или тестирования. Неповеденческие требования часто задаются неконструктивно, что заставляет в процессе формализации привлекать дополнительную информацию, позволяющую реконструировать сценарий их проверки, т. е. привести неконструктивную форму задания требования к конструктивной.

Проверка выполнения требований

Процедура проверки требования — это точная последовательность причин и следствий некоторых активностей (кодируемых действиями, сигналами, состояниями), в результате анализа которой можно утверждать, что данное требование выполнено или нет. Подобная процедура проверки может быть использована в качестве критерия выполнения конкретного требования, т. е. стать т. н. критериальной процедурой. В дальнейшем изложении для критериальной процедуры будем использовать термин последовательность или «цепочка» событий.

Отслеживая в поведенческом сценарии системы (гипотетическом, реализованном в модели или в реальной системе) факты выполнения критериальной процедуры, можно утверждать, что соответствующее требование в анализируемой системе удовлетворено.

Процедура проверки требования (цепочка) формулируется путем задания для всех элементов цепочки следующей информации:

- условий (причин), требующихся для активизации некоторой активности;
- самой активности, подлежащей исполнению при данных условиях;
- следствий — наблюдаемых (измеряемых) результатов исполнения указанной активности.

Для описания причин и следствий используются сигналы, сообщения или транзакции, обычные в коммуникациях инстанций реактивной системы [2], а также состояния переменных в виде значений или ограничений на области допустимых значений. Отслеживая изменения в состояниях, производимых активностями цепочек, можно наблюдать за покрытием соответствующих цепочек.

Проблемы неконструктивного задания требований преодолеваются в процессе разработки процедур проверки выполнения требований на пользовательских или межкомпонентных интерфейсах.

Таким образом, критерием выполнения требований могут служить цепочки, содержащие последовательности активностей и состояний; помимо этого, возможны случаи, в которых критерий выполнения некоторого требования задается не одной, а несколькими цепочками.

Исходные документы, фиксирующие требования к приложению

Формулировка каждого требования в технической документации чаще всего осуществляется на естественном языке и выражается одним из двух способов:

- в виде поведенческого требования, когда из текста требования на естественном языке удается реконструировать сценарий (процедуру) выполнения данного требования;
- в виде неповеденческого требования, задающего состав, структуру или пожелание наличия некоторого свойства без пояснения способа его потребления (способа проверки).

Формализация любых конструктивно заданных требований возможна, как возможен и эффективный автоматизированный анализ требований к программному изделию и реализована в технологии VRS/TAT [3].

В технологии VRS/TAT для высокоуровневого описания модели, в рассматриваемой технологической цепочке используется нотация Use Case Maps (UCM) [4] (Рис.1), а инструменты автоматизации проверки и генерации работают с моделью на языке базовых протоколов (BP) [5].

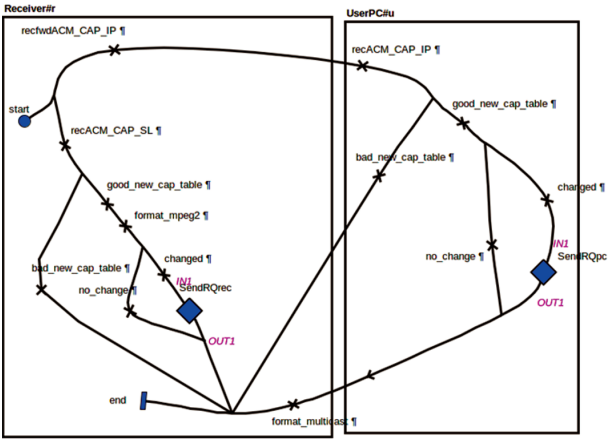


Рис.1. UCM модель двух инстанций Reciver и UserPC

UCM модель (рис.1) содержит описание модели двух взаимодействующих инстанций. Каждый путь на графе от события start до события end

представляет некоторый сценарий поведения. На каждом пути располагается конкретное множество событий (Responsibilities). События на диаграмме обозначены символом ×, а элементы Stub кодирующие вложенную диаграмму символом ♦. В результате каждый сценарий содержит определенную последовательность событий. Множество возможных сценариев задается множеством таких последовательностей.

В таких терминах цепочка определяется как подпоследовательность событий, достаточных для вывода о том, что требование удовлетворено. Путь на UCM диаграмме, содержащий последовательность событий некоторой цепочки, называется трассой, покрывающей соответствующее требование. По трассе можно сгенерировать тесты, необходимые для экспериментального подтверждения выполнимости требования.

Матрица отслеживания

Формализация требований верификационного проекта начинается с разработки матрицы отслеживания TRM [1] — Traceability matrix (на рис.2 TRM для конкретного проекта представлена в виде таблицы). Столбцы Identifier и Requirements содержат идентификатор требования, употребляемый в исходном документе описания требований и полный текст требования, подлежащий формализации. Столбец Traceability содержит цепочки событий, достаточные для проверки выполнимости соответствующего требования, а столбец Traces — трассы или поведенческие сценарии, котороые используются для генерации кода тестов.

Identifier	Requirements	Traceability	Traces
FREQ_GWR.1	Gateway shall transmit ACM_CAP messages repeatedly at an interval of T1 (TBD). This message will carry the ACM Capabilities table.	FREQ_GWR.3	
FREQ_GWR.2	Upon a change in the ACM Capabilities table, the Gateway shall send a new version of ACM_CAP message to the Satellite Terminal.	FREQ_GWR.3	
FREQ_GWR.3	Depending on configuration, ACM_CAP message shall be transmitted either in the MPEG2 private section or in a Multicast message sent across the satellite link.	recACM_CAP_SL _____ recfwdACM_CAP_IP recACM_CAP_IP	FRED_GWR_3-1 FRED_GWR_3-2

Рис. 2. TRM-матрица отслеживания

Например, в строке 3 TRM в столбце Traceability приведены 2 цепочки для покрытия требования FREQ_GWR.3. Для удовлетворения требования достаточно зафиксировать посылку сигнала ACM_CAP по одному из двух возможных сценариев:

FREQ_GWR.3-1: start, **recACMCAP_SL**, good_new_cap_table, format_mpeg2, no_chanes, end

FREQ_GWR.3-2: start, **recfwdACM_CAP_IP**, **recACM_CAP_IP**, good_new_cap_table, format_multicast, end

Следует заметить, что при создании критериальных цепочек строится модель верифицируемой функциональности, в процессе чего вводится много переменных состояния, типов, агентов, инстанций и т.п.

Разработка интегральных критериев покрытия требований

Выше отмечалась характерная черта технологии VRS/TAT — наличие специальных критериев проверки покрытия для каждого требования. Перечислим критерии, относящиеся к требованиям, по степени роста их мощности:

1. Степень покрытия сгенерированными сценариями подмножества событий, использованных в критериальных цепочках.
2. Степень покрытия сгенерированными сценариями подмножества цепочек (состоящих их событий и состояний переменных), не менее чем по одной для каждого требования.
3. Степень покрытия сгенерированными сценариями всего множества цепочек, задающего интегральный (комбинированный из критериев 1–2) критерий покрытия требований.

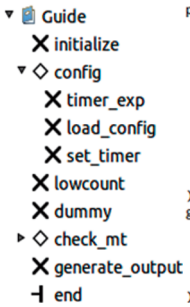
Разработка критериев покрытия должна быть адаптирумой к конкретному проекту [6,12,13]. Критерии должны применяться гибко и могут изменяться в зависимости от условий генерации сценариев.

Генерация сценариев и отбор сценариев, удовлетворяющих выбранному интегральному критерию покрытия

Генерация трасс осуществляется символьным и конкретным трассовыми генераторами STG (SimbolicTrace Generator) и CTG (Concrete Trace Generator), реализующими достаточно эффективные алгоритмы проверки моделей [9,10] (Model Checking). Основной проблемой трассовой генерации является «взрыв» вариантов перебора при генерации сценариев (трасс) из базовых протоколов [1], которые формализуют сценарные события, условия их реализации и соответствующее изменение состояния модели после реализации. Инструментом решения является фильтрация вариантов генерации на основе многочисленных ограничений, специально выбираемых перед циклом генерации трасс.

Существуют универсальные и специальные ограничения. Например, часто используемыми универсальными ограничениями являются ограничения на максимальное количество протоколов, используемых в трассе, и максимальное количество трасс, генерируемых в одном цикле генерации. Специальные ограничения задаются последовательностями событий UCM модели направляющими процесс генерации в интересующем пользователя поведении модели (т. н. гиды — Guides). Используется два этапа для генерации тестовых сценариев по гидам. На первом этапе создаются на

основные гиды, обеспечивающих заданные критерии покрытия поведения системы. На втором этапе гиды в нотации UCM (рис. 3.a) преобразуются в гиды на языке базовых протоколов (рис. 3.b), и под их управлением производится генерация трасс. Важно отметить, что в гидах фиксируются лишь главные контрольные точки поведения, в то время, как трасса, полученная по гиду, содержит детальную последовательность элементов поведения. Такой подход к генерации существенно уменьшает влияние комбинаторного взрыва на время генерации трасс при обходе дерева поведения проектируемой системы.




```
patterns: obj(
    p_Guide:(
        ats_0(initialize#1);
        ats_0(timer_exp_1#1);
        ats_0(load_config_1#1);
        ats_0(set_timer_1#1);
        ats_1(lowcount#1);
        ats_1(dummy#1);
        ats_1(report_req_SADM#1);
        ats_2(generate_output_2#1)
    )
);
guides: obj(
    Number_Of_Traces:1;
    g_Guide:(
        pattern:(p_Guide)
    )
);
```

(a) (b)

Рис. 3. Гиды: (a) в нотации UCM,
(b) на языке базовых протоколов VRS (язык гидов)

Автоматический и ручной процесс создания гидов

Существует два возможных подхода к созданию гидов по высокоуровневому описанию системы на языке UCM: ручной и автоматический.

Автоматический подход позволяет генерировать множество гидов [9,11], обеспечивающих покрытие поведения системы по критерию ветвей [12,13]. Каждый из гидов, содержит информацию о ключевых точках сценария поведения, начиная от начального состояния модели, моделируемого элементом StartPoint, и заканчивая конечным состоянием системы, моделируемым элементом EndPoint. Процесс генерации гидов осуществляется в генератором UCM в MSC [7].

Автоматический подход к созданию гидов можно рассматривать как быстрый способ получения тестового набора, но такой подход не всегда является достаточным. У заказчика и тестового инженера может возникнуть желание проверить особые сценарии поведения системы для проверки, например, какого-либо специфического требования. Такие сценарии разработчик задает вручную, и они могут быть созданы в инструменте Анализа событий UCM (UCM EVA) [7].

В обоих случаях, автоматическом или ручном, могут возникнуть проблемы с осуществлением покрытия гидов тестовыми сценариями.

В автоматическом режиме это связано с тем, что инструменту VRS не всегда удается успешно сгенерировать тестовый сценарий по гидам, т. к. последние создаются на основе потока управления модели и не учитывают особенностей сопряженного с ним потока данных. В то же время трассы, генерируемые на основе потока управления, учитывают изменения значений переменных и соответственно состояний модели.

Поэтому актуальной является задача автоматизации анализа причин непокрытия некоторых гидов трассами и как следствие задача автоматизации отладки гидов, решаемая изменением гидов или UCM модели.

Технологическая цепочка генерации тестовых сценариев

Процесс генерации тестовых сценариев (рис. 4) в технологической цепочке VRS/TAT делится на следующие этапы:

Создание формальной модели системы на языке UCM происходит вручную. Из исходных требований выделяются ключевые события, наблюдаемые в системе. По событиям формируются цепочки, преобразуемые в сценарии поведения системы. Сценарии детализируются, выделяются взаимодействующие объекты системы и структура. В полученную модель вносятся метаданные, необходимые для наделения модели потоком данных. Более подробно методика формализации описана в работе [6].

Созданная UCM модель, с помощью транслятора UCM в BP [3] преобразуется в формальную модель на языке базовых протоколов, который является входным языком для инструментов VRS/TAT. В процессе трансляции осуществляется преобразование элементов и конструкций модели системы с сохранением их семантики, таким образом, в модели из базовых протоколов присутствуют эквивалентное множество поведений UCM модели.

По UCM модели так же осуществляется генерация, с помощью инструмента UCM в MSC, множества гидов, удовлетворяющих критерию покрытия поведения по ветвям. Инструмент генерации осуществляет обход дерева поведения модели, покрывая все возможные ветви.

Если заказчику необходимо проверить какие-либо специфические поведения системы, то он вручную создает требуемое множество в инструменте EVA. Стоит отметить, что гиды создаются в виде последовательности UCM элементов, и для их использования в VRS требуется преобразование в последовательность базовых протоколов. Такое преобразование осуществляется за счет информации получаемой при трансляции в UCM в BP — сохраняется информация о соответствии элементов UCM базовым протоколам.

Затем формальная модель на языке базовых протоколов и гиды поступают на вход трассового генератора VRS, который осуществляет сначала верификацию системы, а затем по верификационной, т. е. доказанной, модели

происходит автоматический обход дерева поведения и построение символьных тестовых сценариев.

После процесса генерации тестовых сценариев необходимо проанализировать покрытие исходных гидов, выявить множество гидов, которые остались непокрыты символьными трассами, и определить причины, по которым они не могут быть покрыты. Такой анализ осуществляется в модуле анализа покрытия инструмента EVA [8].

На основе анализа покрытия вносятся изменения в исходные гиды или UCM модель.

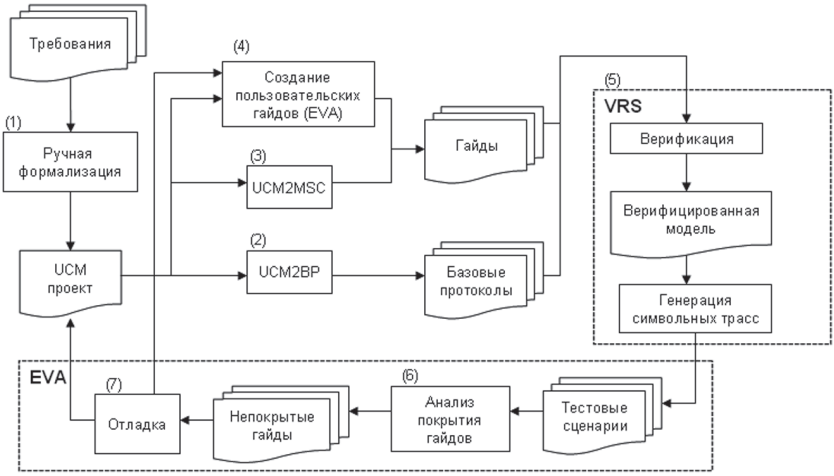


Рис. 4. Процесс генерации тестовых сценариев
с использованием инструмента статического анализа — EVA.

Метод автоматизации отладки гидов

К наиболее частым причинам расхождения относятся: недостаточная детализация гидов и ошибки в последовательности UCM элементов в гиде из-за некорректного использования переменных, идентифицируемой как deadlock на ветвях или ветвлениях.

Рассмотрим метод поиска мест и причин расхождения гида с трассой.

(1) Гиды и, полученные в VRS трассы, представлены в виде MSC диаграмм, отображающих последовательности применения базовых протоколов, поэтому первым шагом алгоритма является отображение имен базовых протоколов на UCM элементы.

(2) Сравнивая гид и трассу в терминах UCM элементов, можно определить последний элемент трассы, удовлетворяющий порядку UCM

элементов, зафиксированному в гиде. Следующий за ним непокрытый элемент гида назовем элементом расхождения.

(3) По непокрытому в символьной трассе элементу расхождения, можно рассмотреть соответствующие ему данные и предусловие.

(4) Далее следует выделить переменные предусловия в отдельный список и проанализировать те места на UCM диаграмме, в которых осуществляется изменения переменных этого списка. Анализ следует вести снизу вверх, начиная с событий ближайших к элементу расхождения.

(5) После обнаружения причины расхождения скорректировать гид или внести изменения в UCM модель.

Предложенную последовательность действий необходимо повторять до тех пор, пока все гиды не будут покрыты трассами.

Проиллюстрируем поиск причины расхождения на простом примере фрагмента телекоммуникационного проекта рис. 5.

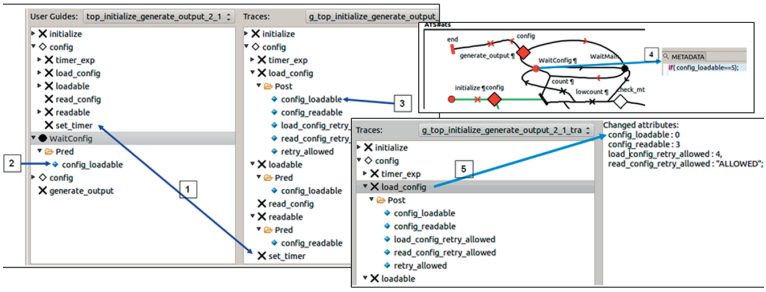


Рис. 5. Выявление причины расхождения гида и трассы по переменным

При выявлении причины непокрытия гида необходимо установить последний элемент совпадения гида и трассы (1) — это элемент **set_timer**. Затем определяется элемент гида, который не может быть достигнут в трассе (элемент расхождения) — это элемент **WaitConfig**. Анализируя его данные (2), выявляем переменную, которая влияет на трассовую генерацию (3) и может являться причиной расхождения это **config_loadable**. По результатам анализа значений этой переменной в UCM модели (4) делаем вывод, что для применимости этого элемента значение должно быть равно 5. Анализ проводится только для тех точек трассы, где использована переменная **config_loadable**. В данном случае такой точкой является элемент **load_config** (5), в котором переменной **config_loadable** присваивается значение — **0**. Тем самым можно сделать вывод, что для создания гида, по которому будет успешно сгенерирована трасса, необходимо изменить присваиваемое значение **5** на элементе load_config или изменить гид.

Получаем выигрыш в сокращении трудоемкости поиска причин ошибок за счет уменьшения количества анализируемых точек, что актуально для больших промышленных проектов, проектируемых в соответствии с рассматриваемой технологией.

Заключение

Результатом работы является усовершенствованная технология, интегрирующая верификацию и тестирование программных проектов, которая обеспечивает:

1. Полную автоматизацию процесса разработки промышленного программного продукта с контролем реализации семантики требований
2. Генерацию модели приложения и символьных поведенческих сценариев, 100% покрывающих поведенческие свойства приложения
3. Автоматическую конкретизацию символьных трасс в соответствии с планом тестирования
4. Высокую автоматизацию процесса разработки и управления качеством программного продукта
5. Результаты применения интегрированной технологии проектирования и тестирования в области разработки беспроводных телекоммуникационных приложений продемонстрировали 26% экономию трудозатрат производства программного продукта.

Литература

1. *Баранов С., Котляров В., Летичевский А*. Индустриальная технология автоматизации тестирования мобильных устройств на основе верифицированных поведенческих моделей проектных спецификаций требований // Труды междунар. науч. конф. «Космос, астрономия и программирование» — СПбГУ, С-Пб. — 2008. — С. 134–145.
2. *Z.Manna, A.Pnueli.*: The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag, 1992.
3. *S.Baranov, V.Kotlyarov, A.Letichevsky, P.Drobintsev*. The technology of Automation Verification and Testing in Industrial Projects. / Proc. of St.Petersburg IEEE Chapter, International Conference, May 18–21, St.Petersburg, Russia, 2005 — pp. 81–86
4. Recommendation ITU-T Z.151. User requirements notation (URN), 11/2008
5. *A. Letichevsky, J. Kapitonova, A. Letichevsky Jr., V. Volkov, S. Baranov, V. Kotlyarov, T. Weigert.* Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications. Proc of ISSRE04 Workshop on Integrated-reliability with Telecommunications and UML Languages (ISSRE04:WITUL), 02 Nov 2004: IRISA Rennes France.
6. *Дробинцев П.Д., Котляров В.П., Черноруцкий И.Г*., Автоматизация тестирования на основе покрытия пользовательских сценариев // Научно-Технические ведомости СПбГПУ. №4 (152), СПб.: Изд-во Политехнического ун-та, 2012. С. 123–126.
7. *И.Ануреев, С.Баранов, Д.Белоглазов, Е.Бодин, П.Дробинцев, А.Колчин, В.Котляров, А.Летичевский, А.Летичевский мл., В.Непомнящий, И.Никифоров, С.Потиенко,*

Л.Прийма, Б.Тютин. Инструментарий поддержки интегрированной технологии анализа и верификации спецификаций телекоммуникационных приложений.// СПИИРАН-2013-№1-27С.

8. *Никифоров И.В., Петров А.В., Котляров В.П.*, Статический метод отладки тестовых сценариев, сгенерированных с помощью эвристик // Научно-технические ведомости СПбГПУ. №4 (152). СПб.: Изд-во Политехнического ун-та, — 2012. — С. 114–119.
9. *А.Колчин, В.Котляров, П.Дробинцев*, Метод генерации тестовых сценариев в среде инсерционного моделирования. // «Управлющие системы и машины», Киев, «Академпериодика», т. 6–2012, С. 42–48
10. *A.A. Letichevsky, J.V. Kapitonova, V.P. Kotlyarov, A.A. Letichevsky Jr., N.S.Nikitchenko, V.A. Volkov, and T.Weigert*. Insertion modeling in distributed system design // Проблеми програмування. — 2008. — С. 13–38.
11. *Летичевский А.А., Колчин А.В*. Генерация тестовых сценариев на основе формальной модели // Проблемы программирования. — 2010. — №2–3. — С. 209–215
12. *Котляров В.П*. Критерии покрытия требований в тестовых сценариях, сгенерированных из повеженческих моделей приложений // Научно-технические ведомости СПбГПУ. — 2011. — Т.6.1(138). — С. 202–207.
13. *Baranov S., Kotlyarov V., Weigert T.* Varifiable Coverage Criteria For Automated Tesdting. SDL2011: Integrating System and Software Modeling // LNCS. — 2012 — Vol.7083 — P. 79–89.

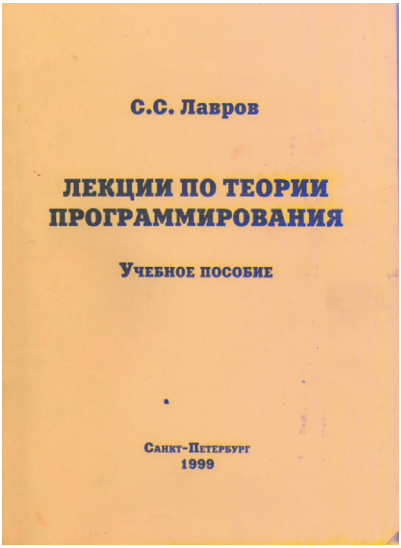
ПОСЛЕДНЯЯ КНИГА СВЯТОСЛАВА СЕРГЕЕВИЧА ЛАВРОВА

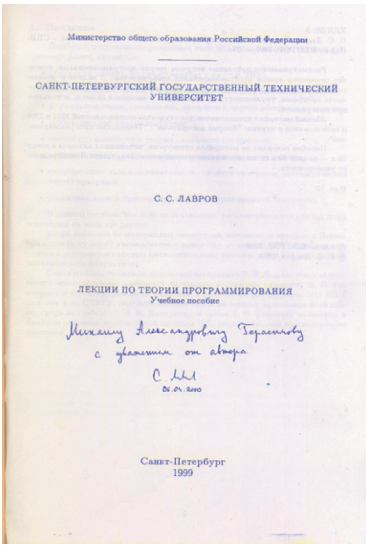
М.А. Герасимов

Санкт-Петербург, каф. Информатики СПбГУ

Данный доклад уже относится скорее к истории компьютерных наук, чем к информатике. Последняя монография С.С. Лаврова содержит обзор последних, на тот момент, достижений в области автоматической верификации и генерации программ. Надо отметить, что большинство идей, изложенных в этой книге, получили свое развитие в самых разных областях современного программирования.

На обложке книги написано — «Учебное пособие», что говорит о том, что книга рассматривалась как учебник для студентов. Поскольку, как мы увидим дальше, для понимания материала книги необходимо знать такие понятия как цепи Маркова, конечные автоматы, исчисления предикатов, понятие обратного вывода, скорее всего этот учебник предназначался студентам 4–5 курсов, специальности «Математическое обеспечение ЭВМ».





О структуре книги

Книга состоит из шести основных разделов (глав), первая из которых — введение, а последняя — послесловие. Таким образом, полновесных глав в книге — четыре:

- Операторные схемы.
- Формализация семантики языков программирования.
- Денотационная семантика составных значений и указателей.
- Денотационная семантика процедур и функций.

ко лет обуче-
как-то: меди-
отив, обычно
ритмические
ого описания

ни охватыва-
ия, тогда как
независимое

пред- и пост-
омежуточных
алось создать
аждую новую
— исходным
фикацию ре-
ессе построе-
из раздела 3.2
олжен был бы
снова идеали-
одукции пока

ществуют си-
пользователь
(т. е. соответ-
тся автомати-
тать и создать
которой есть
всех понятий,
ких программ)

добным систе-

Оглавление

1. Введение . . . 3
2. Операторные схемы . . . 4
   2.1. Оценка трудоемкости алгоритмов . . . 6
      2.1.1. Марковская схема . . . 7
      2.1.2. Метод Кирхгофа . . . 8
   2.2. Доказательство свойств программ . . . 13
   2.3. Завершаемость алгоритмов . . . 17
   2.4. Структурированные схемы . . . 20
   2.5. Экономия памяти . . . 23
3. Формализация семантики языков программирования . . . 27
   3.1. Модельный язык и его операционная семантика . . . 28
   3.2. Исчисление программ . . . 31
   3.3. Состояния и преобразователи состояний . . . 37
   3.4. Целые и логические выражения . . . 38
   3.5. Преобразователи состояний для операторов . . . 40
   3.6. Преобразователи предикатов . . . 46
   3.7. Преобразователи предикатов для операторов . . . 49
   3.8. Обоснование правил деривационной семантики . . . 58
   3.9. Операторные схемы, рекурсия и циклы . . . 61
4. Денотационная семантика составных значений и указателей . . . 67
   4.1. Векторы, записи и ссылки . . . 67
   4.2. Состояния, имена, выражения . . . 70
   4.3. Объявления, генераторы и присваивания . . . 73
   4.4. Блоки . . . 75
   4.5. Простые переменные как указатели . . . 77
   4.6. Динамические типы . . . 79
   4.7. Преобразователи предикатов для присваивания . . . 81
5. Денотационная семантика процедур и функций . . . 86
   5.1. Понятие наименьшей неподвижной точки рекурсивного функционала 86
   5.2. Нерекурсивные процедуры и функции . . . 88
   5.3. Рекурсивные процедуры . . . 93
   5.4. Случай многих процедур . . . 100
   5.5. Преобразователи предикатов для процедур . . . 101
6. Послесловие (За что боролись?) . . . 105

Основные вопросы

Первая глава посвящена вопросам исследования свойств программ на алголоподобных языках программирования. С помощью граф-схем, цепей Маркова и диаграмм Кирхгофа Святослав Сергеевич демонстрирует возможности автоматической оценки сложности программ. Приведены примеры, показывающие возможность извлечения семантических свойств программ только на основе их синтаксической структуры.

Доказываются теоремы, гарантирующие завершаемость алгоритмов, описываемых программами специального вида.

В разделе 2.5, посвященном экономии памяти, вводятся понятия маршрута заданной величины и понятие пучка маршрутов. Рассматриваются так называемые «зацепляющиеся» пучки и области действия локальных и глобальных переменных. Даются алгоритмы, позволяющие применять правило Кирхгофа для определения областей действия переменных.

Модельный язык и денотационная семантика программ

В этой главе вводится определение модельного языка и денотациной семантики этого языка. Вводится операционная семантика программ, определяется исчисление, позволяющее осуществлять автоматический вывод программ с заданными свойствами. Доказываются некоторые важные свойства этого исчисления. Сегодня мы можем видеть развитие этого подхода в системах, основанных на конструктивной логике, таких как система Coq.

В 5-й главе рассматривается денотациная семантика процедур и функций и затрагиваются вопросы так называемых смешаннных (по определению академика Ершова) или «ленивых» вычислений. Отмечается возможность их использования для проверки свойств программ. Сегодня данные подходы нашли свое развитие в языке Haskell.

Задачи, которые требовали решения в ближайшее время

В качестве задач, которые требовали решения в ближайшее время, С.С. Лавровым рассматривались задачи корректного синтеза программ по их формальной спецификации, задачи проверки алгоритмических свойств программ, задачи оптимизации программ с использованием алгебраических методов. Особое внимание уделялось развитию теории автоматического доказательства наличия определенных свойств программ, и созданию специальной системы, способной заменить человека в этом виде деятельности.

Краткие выводы

Основные направления исследований, указанные в книге, остаются актуальными и в наши дни. Некоторые подходы показали свою не перспек-

тивность, другие активно развиваются в современных языках и системах программирования.

До сих пор остаются перспективными направления исследований по извлечению семантики из синтаксической структуры программы, автоматический синтез программ по спецификациям. Таким образом, и сегодня, данная книга остается полезной для теоретиков и практиков программирования.

Литература

1. *Андерсон Р.* Доказательство правильности программ, М.: Мир, 1982.
2. *Лавров С.С.* Лекции по теории программирования. С-Петербург: «Нестор», 1999 — 107с.
3. *Михайлов А.В.* Объектно-ориентированная технология разработки программных систем (UML), 2002. URL: http://gis-2000.narod.ru/books.html
4. *Новоженов Ю.В.* Опыт реинжиниринга объектно-ориентированного комплекса программ с применением CASE-средства Rational Rose и SILVERRUN.
5. *Рамбо Дж.*, UML 2.0. Объектно-ориентированное моделирование и разработка — М.:2007.-544с
6. *Khusidman V.* Business Rules Discovery from Existing Applications // Business Rules Journal, Vol. 8, No. 10. Business Rule Community, 2007. URL: http://www.BRCommunity.com/a2007/b366.html

МИКРОКОНТЕКСТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД К КОМПОНЕНТНОМУ ПОСТРОЕНИЮ ЯЗЫКОВ

В.В. Прохоров

д. ф.-м. н., проф. УрФУ, vpro@vidicor.ru

Аннотация: Рассматривается подход к разрешению проблемы противоречия между богатством средств языка информационных описаний и его простотой, компактностью. Подход основан на рассмотрении языка, программных средств/оборудования как сложной системы с применением соответствующей методологии. Подход направлен на решение проблемы организации языков представления знаний, человеко-компьютерного взаимодействия, программных и аппаратных средств.

Введение

Для описания объектов и задач в какой-либо области желательно применять языки, максимально адекватные этой области как по содержанию, так и по форме. При этом нежелателен слишком большой разрыв (как по *внутренней сущности*, так и по *внешней форме*) между принятым в частной прикладной области уже сложившимся языком, естественным для специалиста в этой области, и специальным формальным языком представления знаний. Специалисту важно иметь возможность использовать в работе не специально введённые новые представления для описания предметной области, а те же самые понятия и концепции, которые используются в этой области традиционно. Эффективная реализация такой возможности помогает максимально сохранить семантику уже существующего мира понятий и перенести её в модель представления знаний.

Для описания каждой из частей общей модели могут исходно применяться весьма различные средства: дифференциальные уравнения, логические, алгоритмические описания, семантические сети, продукции, структурные диаграммы, фреймы, описания структур данных, описания пространственных сцен, описания электронных схем, музыки и др. Это многоязычие создаёт трудности при описании сложной системы *в целом*. Использование *единственной* описательной модели для сложной системы вынуждает представлять большую долю информации в форме, *не адекватной природе описываемых сущностей*.

При этом проблема «наилучшего языка» существенно связана с противоречием между его богатством и компактностью: язык должен иметь *высокую описательную мощь*, но в то же время по количеству конструкций должен быть достаточно простым в освоении.

При использовании компьютерных технологий вышесказанное предстаёт как проблема *дистанции* между исходными моделями, описывающими не-

обходимые сущности предметной области в привычной для этой области форме, и моделями, которые могут быть непосредственно «поняты» *компьютерной средой*. Преодоление данной проблемы всегда являлось одной из центральных задач компьютерных технологий, начиная с зарождения этих технологий и по настоящее время. Заметим, что любой компьютерный язык в первую очередь рассматривается здесь не как средство задания процесса функционирования компьютера, а как *средство представления некоторых информационных моделей*, используемое некоей «виртуальной машиной».

В работе обсуждается *иерархическая микроконтекстная мультипарадигменная компонентная технология (микроконтекстно-ориентированый подход)* [1 – 7] — взгляд на построение языков информационных описаний, организацию программных и аппаратных средств и процесс создания программных продуктов, обобщающий концепцию модульного проектирования на «языковую ситуацию». В предлагаемом решении каждый из иерархически связанных модулей-микроязыков (подъязыков) является «клеем», связывающим субмодули своими собственными средствами (включая и внешние формы представления), независимо от средств, используемых другими модулями структуры (и для человеко-компьютерного взаимодействия в целом, а не только для области «программирования»).

Традиционные подходы к решению противоречия между простотой языка и богатством его средств

В современной практике встречается ряд подходов к решению противоречия между простотой языка и богатством его средств: однопарадигменный подход, мультипарадигменный подход, среда, расширяемая пользователем, … Рассмотрим эти подходы более подробно.

Однопарадигменный подход

Однопарадигменный подход предлагает использовать некоторый универсальный, независимый от области применения, язык, базирующийся на некоторой зафиксированной для этого языка *одной главной парадигме* (например, алгоритмической). Единственность базовой парадигмы обеспечивает небольшое количество базовых элементов такого языка. Это близко к идее «RISC» (Reduced Instruction Set Computer) для микропроцессоров.

При работе с узким классом информационных моделей проблема человеко-компьютерного взаимодействия решается наиболее успешно при использовании специализированной информационной среды, то есть языка и соответствующей «виртуальной машины», ориентированных именно на этот класс.

Однако ситуация меняется, если решаемая проблема сложна и включает много разнородных частей, которым наиболее адекватны информационные

описания различной природы. В этом случае возможно использование компактных монопарадигменных универсальных средств («притягивание» информационной модели к имеющейся информационной среде). В этом случае для конкретной области необходимо свести каждый объект, с которым имеют дело, к выбранной единственной парадигме (или к небольшому количеству парадигм) и единственному языку. Чтобы применять этот подход, требуется остановиться на выборе некоторой универсальной парадигмы (например, алгоритмической, функциональной, логической и т. п.), выбрать соответствующий компьютерный язык и соответствующую компьютерную среду, поддерживающую работу на этом языке. При этом необходимо применить определённые научные и прикладные наработки (сведению разных видов моделей к алгоритмической парадигме посвящены целые области математики) и искусство, чтобы сводить задачи из различных областей к моделям в терминах выбранной парадигмы (языка). Самый популярный базис этого подхода — алгоритмическая парадигма.

Основные недостатки подхода монопарадигменных универсальных средств:

- использование монопарадигменного подхода требует трудоёмкой работы по сведению различных классов исходных моделей представления к неадекватной единой форме (например, запись музыки или вёрстка полиграфического продукта средствами универсального алгоритмического языка),
- работа с единственной парадигмой деформирует сознание, затрудняя работу с представлениями в других парадигмах.

Мультипарадигменный подход

Мультипарадигменный подход характеризует «расширение» монопарадигменной универсальной системы вводом новых элементов, удобных для работ, которые исходно были нехарактерны для природы применяемого языка, облегчающих его использование и соответствующих некоторым другим парадигмам. При дальнейшем развитии этот подход приводит к формированию большого *мультипарадигменного языка* с большим количеством включенных в него парадигм. Этот подход подобен подходу «CISC» (Complex Instruction Set Computer) для микропроцессоров. Как правило, структура связи элементов мультипарадигменного языка жёстко зафиксирована разработчиками такой системы. В соответствии с этим подходом необходимо разработать огромное разнообразие специальных сложных языков (и систем) для различных областей применения, объединяя основные метафоры в различных комбинациях. Мультипарадигменные системы могут оказаться узкоспециализированными и достаточно сложными в изучении, использовании и создании поддерживающих их программных сред.

Некоторые из недостатков мультипарадигменного подхода:

- ориентированность универсальных систем на использование, как правило, какой-то одной формы внешнего представления информационных моделей;
- невозможность включить в универсальную систему все, что кому-то когда-то может понадобиться, в удобной форме;
- даже когда в язык включено не «все для всех», он оказывается громоздким как для разработки, так и для овладения;
- при множественности компонент «интегрированной системы» некоторые из них оказываются реализованы очень удачно, другие — менее удачно. Но все они составляют единый неделимый комплекс и как правило пользователю системы никакие её части заменить невозможно;
- создание таких систем весьма дорого для потребителя;
- программистское качество отдельных компонент может оказаться не столь высоким, как у массовых универсальных систем;
- такие системы, будучи менее универсальными, ставят потребителя в весьма зависимое положение: система позволяет манипулировать лишь в тех сферах и в том виде, как это предполагал её разработчик.

Среда, расширяемая пользователем

В случае среды, расширяемой пользователем, информационная система может расширяться самим пользователем. Яркий пример – концепция языков Forth или Lisp в программировании. Чаще предусматривают менее радикальные возможности по операционному расширению среды для адаптации универсальных средств к решаемой задаче. Например, предоставление возможности создания пользователем библиотеки процедур в алгоритмическом языке, макросов в текстовом или графическом редакторе, объектов в объектно-ориентированном программировании.

При этом пользователь, работая в произвольной среде, по сути порождает в процессе работы проблемно-ориентированную мультипарадигменную языковую среду, расширяя базовые средства. Однако работа как с базовыми объектами, так и с элементами, построенными при расширении, производится независимо от их внутреннего содержания в рамках одной формы внешнего представления (например, текстового) подчас плохо соответствующего природе описываемых сущностей, что является значительным недостатком.

Изложенное относится к проблеме представления не только компьютерно-ориентированных моделей. Однако в областях *человеко-компьютерного* взаимодействия (а также и — ещё более узко — для области языков *программирования*) к общим проблемам представления модели добавляются проблемы *обработки* описания, связанные как с программным обеспечением, так и с аппаратными компьютерными средствами.

В целом для упомянутых подходов очевидно **противоречие между богатством изобразительных возможностей языка и его простотой и гибкостью**. Это противоречие препятствует попыткам создания единого «идеального языка».

Представляется, что решение этого противоречия даёт только использование *составных* иерархических компонентных информационных описаний и сред.

Язык как сложная система

Как было отмечено выше, при работе со сложными явлениями используемый язык явно (в своих конструкциях) или неявно (за счёт расширяемости языка) становится мультипарадигменным, отражая многосторонность описываемых сущностей, композицией неких более простых средств.

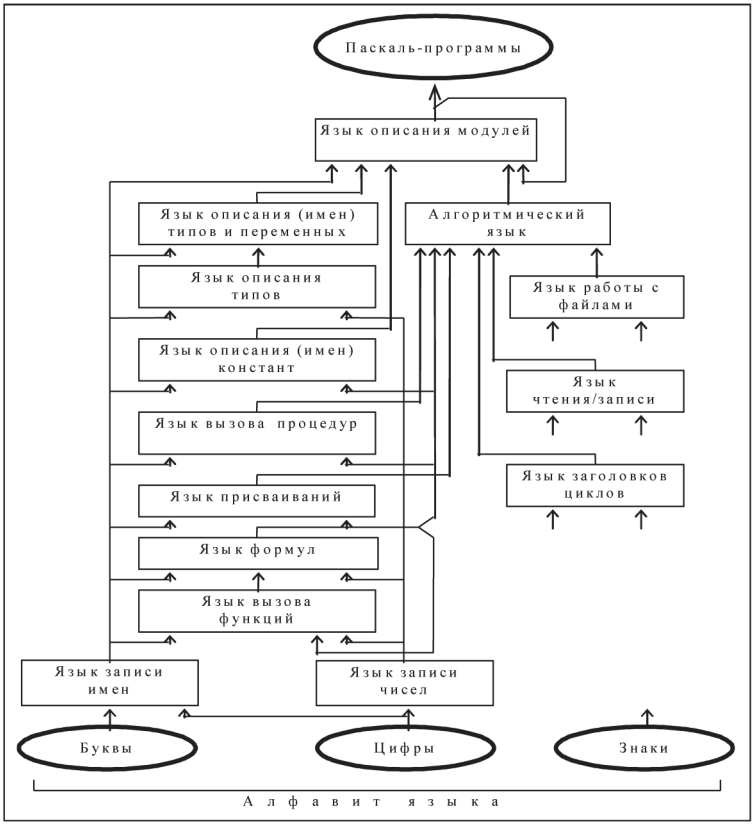


Рис. 1. Препарирование сложного языка на подъязыки

Представляется весьма полезным проведение анализа любой универсальной языковой среды с попыткой выделения достаточно самостоятельных языковых слоёв, имеющих разную синтаксическую и концептуальную основу.

В качестве примера проведём разбиение языка Pascal на структуру подъязыков (Рис. 1) и выделим конструкции на разных подъязыках в некоторой программе на языке Pascal (Рис. 2).

Из «препарирования» даже такого простейшего языка, как Pascal, видно, что он является мультипарадигменным комплексом. Называть его алгоритми-

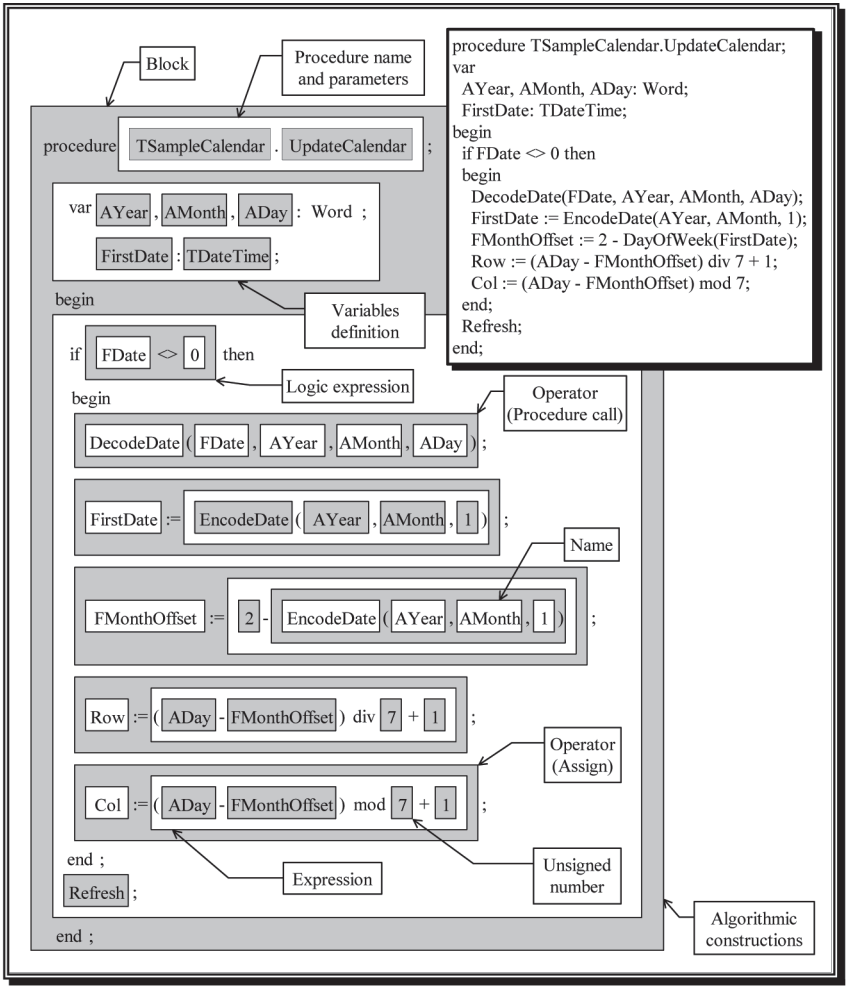


Рис. 2. Подъязыки в Pascal-программе

ческим [монопарадигменным] языком неправомерно. В предлагаемой нами модели Pascal лишь ***включает*** в себя алгоритмический подъязык наряду с рядом других, неалгоритмических (языком формул, языком описания типов и др.).

Аналогичное «анатомирование» можно провести и для любых других нетривиальных систем и их языков информационных спецификаций, будь то текстовый или графический редактор, системы программирования, системы управления базами данных или любые другие системы произвольного назначения.

Заметим, что само «анатомирование» произвольных систем и языков с позиций сложных систем полезно как для лучшего понимания сути конкретной системы и языка, так и для выделения многообразия, «коллекции» «ортогональных» используемых правил композиции/декомпозиции.

Если нарисовать диаграммы, подобные схеме Рис.1, для множества различных языков, то можно увидеть, что многие микроязыки (например, для

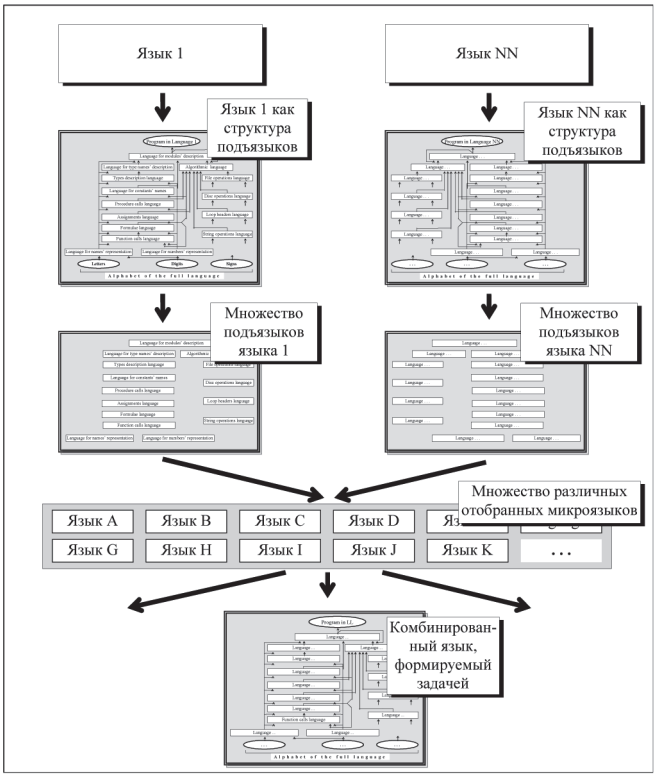


Рис. 3. Демонтаж сложных языков, создание набора различных микроязыков, сборка мультипарадигменного языка для некоторой специфической задачи

алгоритмических конструкций) встречаются в огромном многообразии языков (с некоторыми различиями). Так что «анатомирование» весьма полезно и для выявления наилучших экземпляров конкретной реализации каждой группы правил. Такой анализ подтверждает и то, что количество существенно различных используемых классов правил композиции/декомпозиции значительно меньше, чем количество различных существующих языков (по оценкам в настоящее время известно около 2 500 языков программирования).

Идея «анатомирования» сложных языков влечёт идею микроконтекстно-ориентированного подхода.

В обычном подходе элементы языка жёстко связаны при проектировании. А что если бы была возможность компоновать язык самостоятельно из ассортимента независимых микроязыков?

Микроконтекстно-ориентированный подход

Идея «анатомирования» существующих информационных систем подводит к идее адаптивного конструирования, когда компоненты языка не являются жёстко встроенными в систему, а имеют самостоятельность, могут выбираться по обстоятельствам из множества, не связанного именно с какой-то монолитной системой, то есть среда (и язык, и обеспечивающие средства) оказывается подобной конструктору «Lego». Предлагаемый подход к построению информационных систем — альтернатива концепции универсальных больших языков с большими средами, «интегрированных систем» с полностью определяемым производителем системы множеством компонент и их связей.

Концепция предполагает, что из всех видов информационных работ могут быть выделены некоторые «атомарные» типы и слои, соответствующие некоторым достаточно независимым *компактным* классам информационных моделей. Причём базовые типы работ должны выделяться не только в «горизонтальном» направлении, но прежде всего – в «вертикальном», как элементы иерархической структуры. Каждый тип работ предлагается рассматривать, как группу правил, позволяющих из некоторых объектов синтезировать более сложные, если рассматривать технологию «снизу вверх» или же представлять сложный объект в виде композиции более простых, если рассматривать технологию «сверху вниз». Каждый из базовых языков спецификаций трактуется, как пара {<набор базовых элементов>, <набор средств композиции/декомпозиции>}. В соответствии с этим можно сгруппировать встречающиеся базовые элементы и наборы средств композиции/декомпозиции на классы. Эта идея отражена на Рис.3.

Мы говорим в основном о типах наборов средств композиции/декомпозиции, полагая, что для каждого из этих типов наборы базовых элементов подразумеваются.

Сборка микроязыков

Дополнительно к описанному выше можно, прежде всего, ***освободить связи между подъязыками***, выделенными при анатомировании различных языков. Тогда эти подъязыки можно рассматривать, как ***сырье, для сборки новых языков***. Это позволяет *динамически* строить *ориентированный на задачу* язык представления, который наилучше всего подходит для специфической проблемы. Кроме того, *каждый из подъязыков мог бы обладать наилучшей формой внешнего представления*, например, могут оказаться смешаны и текстовые, и визуальные подъязыки.

Формула технологии компонентных языков

Содержание технологии предполагает следующее:

1. Всякое информационное описание, исключая базисное, рассматривается как иерархическая структура, получаемая из более простых информационных описаний (слоёв), каждое из которых построено средствами своей информационной системы.
2. Каждая из используемых ИС ориентирована на весьма специализированный, узкий класс операций, множество ее правил весьма компактно.
3. Для каждой ИС имеются:
   - независимая внутренняя концепция языка описаний,
   - независимая форма внешних представлений спецификаций.
4. Существование слоя описания разделяется на фазы, такие как:
   - разработка описания-модели,
   - ввод/редактирование,
   - вывод на экран и на печать,
   - интерпретация и/или конвертирование,
   - компиляция, и др.
5. Для каждого слоя и каждой фазы имеются независимые, вообще говоря, оборудование и программная поддержка. Эти программное обеспечение и оборудование могут быть созданы независимо, иметь свою независимую фирму-изготовитель.
6. Существуют соглашения, поддерживающие многоцелевое использование каждой информационной системы, неединственность пар слоёв, для которых указанные в (5) компоненты можно использовать совместно.
7. Имеется возможность самостоятельного комплектования пользователем или системным интегратором иерархической программной системы из программных и аппаратных средств поддержки языковых слоёв (5).

Стадии существования описания и поддерживающего программного обеспечения

Можно рассмотреть различные «стадии существования» описания знания, включая связанные с использованием компьютерной техники, например:

- разработка модели явлений;
- создание и изменение (редактирование) формального описания с использованием компьютерной среды;
- компиляция или другое преобразование описания;
- компьютерное моделирование или интерпретация описания («выполнение»);
- уничтожение части описания на подъязыке;
- документирование (печать) описания;
- действия некоторого редактора: сохранение подописания, вставка от буфера и т. д.

Если мы имеем дело со «сборным языком», то естественно желать, чтобы и программное обеспечение и аппаратные компьютерные средства, поддерживающие подготовку описания и различные виды работы с ним, были тоже сборными.

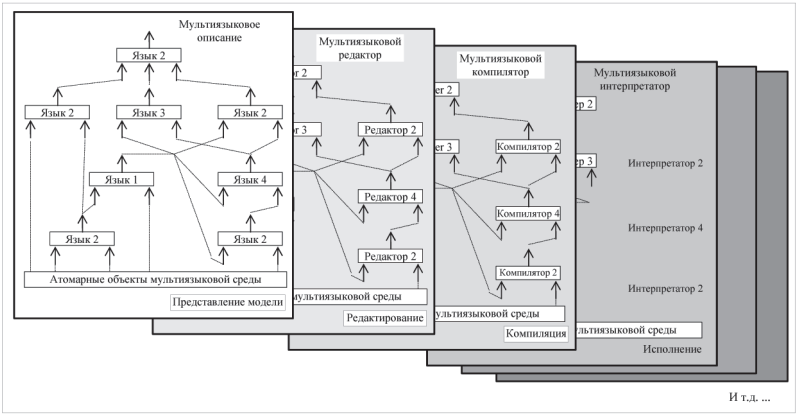


Рис. 5. Иерархия программных средств для разных стадий существования описания

Таким образом, было бы превосходно, если бы для каждого микроязыка имелось программное обеспечение, поддерживающее каждую стадию существования описания, причём компоненты этой программной среды могли бы работать совместно в различных комбинациях, адаптивно формируя ориентированные на задачу редактор, компилятор, интерпретатор и т.д.

Следовательно, чтобы определить некоторый подъязык для компьютерного использования, мы должны определить:

- программу, которая должна помочь, если пользователь захочет редактировать фрагмент на этом подъязыке,
- программу, которая должна помочь, если компилятору вышележащего слоя потребуется компилировать фрагмент на этом подъязыке,
- программу, которая должна помочь, если при выполнении вышележащего слоя потребуется интерпретировать фрагмент на этом подъязыке и т. д.

Программные средства должны быть согласованы, чтобы было возможным их совместное функционирование. Подчеркнём, что работа на каждой фазе выглядит, как работа в единой среде, работа с единым языком. Так, скажем мы могли бы редактировать иерархически вложенные части описания, как подъязыковые фрагменты одной «многоязыковой» программы (как это происходит при работе с «составными документами»).

Уровни в иерархии моделей (параллели с формулой «модель, алгоритм, программа»)

На Рис. 4 изображена возможная иерархия описаний для сложной системы.

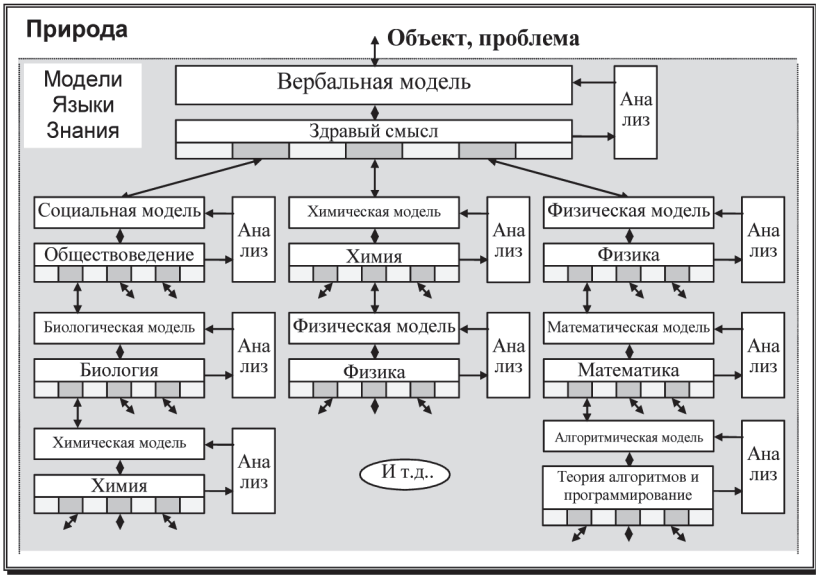


Рис. 4. Языковая иерархия при описании сложной системы.

Достоинства и проблемы реализации технологии

Достоинства использования иерархической микроконтекстной технологии представляются следующими:

- каждый элемент сложной информационной работы выполняется в микросреде, наиболее подходящей для работы как по языку, так и по интерфейсному оборудованию;
- возможность независимого выбора производителей различных подсред и их замены обеспечивает высокое качество компонент комплекса;
- программное обеспечение, поддерживающее работу в каждой из подсред (диалоговый монитор или редактор, интерпретатор или компилятор) весьма компактны и поэтому надёжны и дёшевы;
- каждая из подсред проста в работе благодаря её компактности;
- единый подход к различным информационным концепциям (т. е. в сфере программирования снимается концептуальное различие между «алгоритмическими», «функциональными», «логическими» языками, между «программированием» и «использованием готовых программ»);
- даже весьма небольшой набор различных подсред порождает значительные многообразие и мощность конкретных сред, иерархически формируемых этим множеством;
- обеспечивается максимальная возможность утилизации ранее разработанных элементов («reuse»).

Ряд черт данной технологии в той или иной степени проявляются в современных индустриальных технологиях.

Литература

1. *Прохоров В.В.* π-схемы — язык графического представления алгоритмов. // «Кибернетика и системный анализ», №2, 1992 (ISSN 1060-0396). — С.93 – 107.
2. *Prokhorov, V.* PYTHAGORAS: Multienvironment Software. // In: B. Blumenthal, J. Gornostaev, and C. Unger, Eds. Human-Computer Interaction. Lecture Notes in Computer Science (LNCS), Vol.1015. Springer Verlag, Berlin, Germany, 1995. (ISBN 3-540-60614-9). — P.135 – 148.
3. *Prokhorov, V.* On Graph Representation of Syntax Definitions. // In: J.G. Chen, Ed. Expert Systems Applications & Artificial Intelligence. Technology Transfer Series. Gournay-Sur-Marne, France: IITT International, 1995 (ISBN 2-90766931-1). — P.53 – 58.
4. *Прохоров В.В.* О микроконтекстном подходе к построению языков представления знаний и человеко-компьютерного взаимодействия. // Известия РАН. Теория и системы управления, 1997, №5 (ISSN 1064-2307, 1555-6530). — С.5 – 16.
5. *Prokhorov, V.* Computational Portal: Remote Access to High-Performance Computing. // In: V. Malyshkin, Ed. Parallel Computing Technologies. Lectures Notes in Computer Science (LNCS). Vol.2127. Springer Verlag, Berlin, Germany, 2001. P.308 – 313.

6. *Прохоров В.В.* Комплекс интернет-медиасредств на базе компонентной π технологии // Алгоритмы и программные средства параллельных вычислений. Вып.6. — Екатеринбург: ИММ УрО РАН, 2002. С.289 – 356.
7. *Prokhorov, V.* On Microcontext Representation of Knowledge and Programs. // В. М. Кормышев, ред. Современные компьютерные и информационные технологии. Сборник трудов международной научной Российско-Корейской конференции. Екатеринбург: УрФУ, 2012. — С.9 – 32.

АЛГЕБРАИЧЕСКИЕ БАЙЕСОВСКИЕ СЕТИ И РОДСТВЕННЫЕ МОДЕЛИ ЗНАНИЙ С НЕОПРЕДЕЛЕННОСТЬЮ*

Тулупьев А. Л.

д.ф.-м.н., доц., зав. лаб. теоретических
и междисциплинарных проблем информатики, профессор;
Санкт-Петербургский институт информатики и автоматизации РАН,
Санкт-Петербургский государственный университет,
e-mail: alt@iias.spb.su; alexander.tulupyev@gmail.com

Введение

Вероятностные графические модели (ВГМ, в том числе и алгебраические байесовские сети — АБС [14, 15], байесовские сети доверия — БСД [3], марковские сети — МС [2]) являются одной из признанных теоретических основ информатики: достаточно вспомнить, что премию Тьюринга за 2011 год АСМ решилы вручить проф. Дж. Перлу, основоположнику теории БСД [1]. Вместе с тем, ВГМ допускают рассмотрение с совершенно разных позиций. С точки зрения теории вероятностей и математической статистики, они представляют собой совокупность случайных элементов с разреженной сетью зависимостей. С точки зрения интеллектуального анализа данных, вероятностные графические модели — это способ представить в обозримом, понятном для неподготовленного математически пользователя виде знания, извлеченные из обучающей выборки и, возможно, из других источников. С точки зрения искусственного интеллекта и мягких вычислений, ВГМ — это модели сложных систем знаний с неопределенностью, снабженные алгоритмами вероятностного или логико-вероятностного вывода в условиях различных видов неопределенности, дефицита информации. В прикладных аспектах психология и другие «когнитивные науки» относятся к ВГМ как к одному из видов когнитивных моделей, а теория ВГМ в ряде случаев служит основой исследовательского инструментария когнитивных наук, в частности, в тех сферах, когда приходится увязывать когнитивные и поведенческие стороны объетов исследования.

Цель доклада — краткое описание преимущественно алгебраических байесовских сетей, дополненное описанием байесовских сетей доверия и марковских сетей, а также реферативное упоминание актуальных областей приложений теорий ВГМ.

---

* Работа получила частичную финансовую поддержку в форме гранта РФФИ, проект № 12-01-00945-а.

Алгебраические байесовские сети

С формальной точки зрения, алгебраическая байесовская сеть (рис. 1, а) — это совокупность идеалов конъюнктов, в которой каждому конъюнкту приписана скалярная либо интервальная оценка вероятности его истинности [14, 15].

Идеал конъюнктов рассматривается как логико-вероятностная модель фрагмента знаний, сама сеть — как логико-вероятностная графическая модель баз фрагментов знаний с неопределенностью. Усиленный акцент на выделение фрагментов знаний не случаен: алгебраические байесовские сети, как и другие родственные ей вероятностные графические модели, получают преимущество за счёт использования принципа декомпозиции. В данном случае он проявляется в том, что сложную систему знаний можно разбить на фрагменты относительно небольшого размера. В математических моделях таких фрагментов знаний можно использовать алгоритмы логико-вероятностного вывода, как правило, не сталкиваясь с вопросами недостижимого уровня сложности вычислений: размеры фрагментов знаний изначально предполагаются ограниченными настолько, насколько требуется для реализуемости вычислений, необходимых для локальных алгоритмов логико-вероятностного вывода.

Однако результаты локальных вычислений необходимо распространять «вовне», в соседние фрагменты знаний, а через них — далее, снова переходя к их соседям. Это справедливо как в отношении алгоритмов проверки и поддержания непротиворечивости, априорного вывода, так и в отношении алгоритмов апостериорного вывода — распространения влияния свидетельств. Таким образом, встает вопрос о формализации и исследовании глобальной структуры алгебраических байесовских сетей, отражающей возникающие связи между фрагментами знаний. В настоящее время в теории алгебраических байесовских сетей в качестве основной глобальной структуры рассматривается вторичная структура — она представляется в виде графа смежности, который обладаем свойством магистральной связности — тем свойством, на которое явным или неявным образом опираются все приемы распространить результаты локальных вычислений по сети. Однако более глубокое изучение свойств вторичной структуры алгебраических байесовских сетей привело к выделению их третичной и четвертичной структур, а также протоструктуры, которая оказалась полезна при рассмотрении алгебраических байесовских сетей и сопоставляющихся им графов смежности с точки зрения теории гиперграфов [16, 17].

Возвращаясь к роли принципа декомпозиции в теориях алгебраических байесовских сетей и родственных моделей, следует отметить, что глобальные структуры таких сетей предполагаются достаточно разреженными — только при этих условиях удается эффективно (с приемлемой вычислительной сложностью) распространить влияние локальных вы-

числений по всей сети. Таким образом, принцип декомпозиции является не только преимуществом вероятностных графических моделей, но также задает определенные границы их применимости. Хотя эта проблема не будет подниматься в докладе, отметим, что развивается отдельное направление, которое стремиться найти замену «плотно заполненным» глобальным структурам в виде близких (в каком-то заданном смысле) им разреженных структур.

За счет удачно подобранной индексации конъюнктов — элементов идеала конъюнктов, а также квантов — конъюнкций, построенных на атомами, вошедшими в идеал конъюнктов, причем каждый атом входит в конъюнкцию либо положительно, либо с отрицанием (пример векторов вероятностей соответствующих элементов, приведенных в порядке индексации, представлен на рис. 1, в), все основные задачи локального логико-вероятностного вывода удалось свести к матрично-векторным уравнениям и неравенствам, либо сериям задач линейного программирования, которые задаются с использованием упомянутых матрично-векторных уравнений и неравенств.

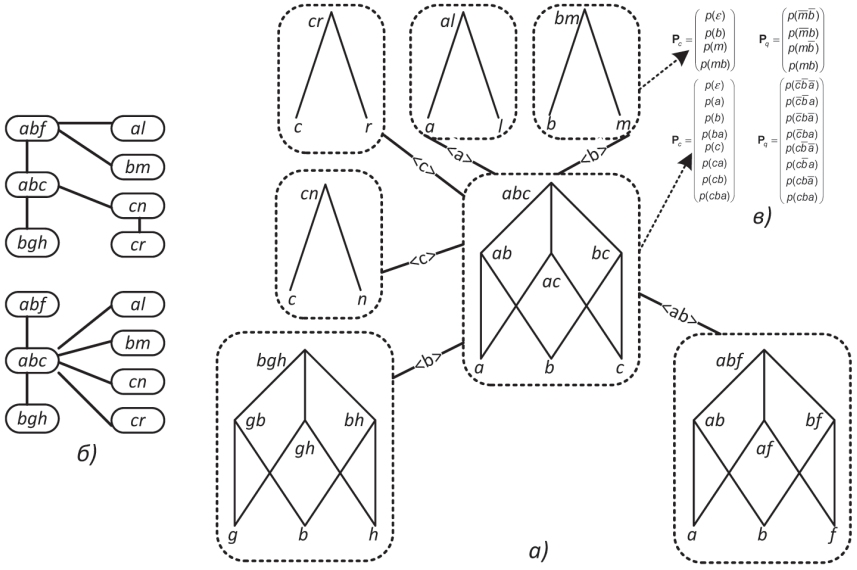


Рис. 1. Графические представления АБС, векторы вероятностей конъюнктов и квантов

В частности, проверка непротиворечивости задания скалярных оценок вероятностей на идеале конъюнктов сводится к проверке следующего не-

равенства (мы используем обозначения, принятые в теории решения экстремальных задач):

$$\mathbf{I}_k\mathbf{P}_c \geqslant \mathbf{0}_k,$$

где k — число атомов, вошедших в идеал конъюнктов, $\mathbf{0}_k$ — вектор-столбец соответствующей размерности, состоящий из нулей, а матрица

$$\mathbf{I}_k = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}^{[k]},$$

причём $[k]$ в показателе обозначает степень Кронекера квадратной матрицы.

Другим примером удачного применения матрично-векторного языка в теории алгебраических байесовских сетей служат формулы, которые используются в апостериорном выводе. Рассмотри ситуацию, когда во фрагмент знаний со скалярными оценками поступает детерминированное свидетельство. Детальные определения можно найти в [12, 15], но в текущем контексте можно рассматривать детерминированное свидетельство как конъюнкцию части атомов из фрагмента знаний, причем некоторые атомы могут входить положительно, а остальные — с отрицанием. Пусть двоичная запись числа i задает бинарный характеристический вектор атомов, которые вошли в свидетельство с положительным означиванием, а двоичная запись числа j — бинарный характеристический вектор атомов, которые вошли в свидетельство с отрицанием. Тогда вероятность появления свидетельства над рассматриваемым фрагментом знаний вычисляется как

$$p\left(\langle i,j\rangle\right) = \left(\mathbf{T}^{\langle i,j\rangle}\mathbf{P}_c\right)[0],$$

а вектор условных (т. е., в данном случае, апостериорных) вероятностей элементов фрагмента знаний вычисляется по формуле

$$\mathbf{P}_c^{\langle i,j\rangle} = \frac{\mathbf{T}^{\langle i,j\rangle}\mathbf{P}_c}{\left(\mathbf{T}^{\langle i,j\rangle}\mathbf{P}_c\right)[0]},$$

где $[0]$ обозначает выборку значения самого верхнего элемента получающегося вектор-столбца (из-за использования побитовых вычислений индексация элементов в векторе начинается с нуля), определены матрицы

$$\mathbf{J}_k = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad \mathbf{H}^{\langle i,j\rangle} = \bigotimes_{t=k-1}^{t=0} \widetilde{\mathbf{H}}_t^{\langle i,j\rangle},$$

$$\widetilde{\mathbf{H}}_t^{\langle i,j\rangle} = \begin{cases} \mathbf{H}^{+}, \text{ если } k\text{-й атом в свидетельстве положителен,} \\ \mathbf{H}^{-}, \text{ если } k\text{-й атом в свидетельстве с отрицанием,} \\ \mathbf{H}^{\circ}, \text{ иначе,} \end{cases}$$

$$\mathbf{H}^{+}=\begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix},\quad \mathbf{H}^{-}=\begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix},\quad \mathbf{H}^{\circ}=\mathbf{H}^{+}+\mathbf{H}^{-},$$

наконец,

$$\mathbf{T}^{\langle i,j\rangle}=\mathbf{J}_k\mathbf{H}^{\langle i,j\rangle}\mathbf{I}_k,$$

а символ $\otimes$ обозначает прямое (тензорное) произведение матриц.

Формализация описания операций локального логико-вероятностного вывода в АБС существенно облегчила их алгоритмизацию, а также упростила исследования свойств получаемых результатов. В частности, были исследованы вопросы чувствительности результатов априорного вывода и ненормированного результата апостериорного вывода. Вместе с тем, следует признать, что даже в сфере локального логико-вероятностного вывода остаются неисследованным ряд вопросов и нерешенным ряд проблем. В частности, не решена еще проблема оценки чувствительности нормированных результатов апостериорного вывода, не решен вопрос о возможности или способе алгоритмизации апостериорного вывода точных апостериорных оценок (вместо накрывающих оценок, способ получения которых известен и сведен к решению серии задач линейного программирования). Эти решения позволили бы предъявлять обоснованные требования к точности априорных оценок в алгебраических байесовских сетях исходя из желаемой точности апостериорных оценок.

Отметим, что сейчас ведутся работы по описанию видов глобального логико-вероятностного вывода на матрично-векторном языке, в частности — описания процесса распространения виртуального свидетельства. Такое описание снова, в свою очередь, позволило бы не только упростить и улучшить алгоритмизацию ряда видов вывода, но и исследовать свойства получающихся результатов методами некоторых разделов линейной алгебры и теории экстремальных задач.

Особняком стоит комплексная проблема алгоритмизации машинного обучения (автоматического обучения) алгебраических байесовских сетей. Погружаясь в более широкий контекст, отметим, что такие же комплексные проблемы стоят и в теориях всех вероятностных графических моделей. Хотя можно сформулировать общую задачу машинного обучения вероятностных графических моделей и выделить похожие подзадачи, не следует упускать из виду, что в теории каждого класса вероятностных графических моделей декомпозиция общей задачи машинного обучения будет индивидуальна и более богата. Она, в конечном счете, сведется к разным системам подзадач, ибо в их постановку существенный вклад вносят собственные особенности рассматриваемого класса вероятностных графических моделей.

Оставаясь на самом высоком уровне иерархии вероятностных графических моделей, комплексную проблему машинного обучения вероятност-

ной графической модели можно формулировать следующим образом: «по совокупности имеющихся данных, а также с учетом дополнительно поступивших знаний и предметной области построить вероятностную графическую модель, наиболее удачно (в каком-то заданном смысле, допускающем подходящую формализацию) отражающую поступившие данные и знания либо наиболее удачно согласованную с поступившими данными и знаниями». Даже в самой общей постановке в комплексной проблеме машинного обучения вероятностных графических моделей можно увидеть проблему построения глобальной структуры модели и проблему оценки параметров такой модели (тех вероятностей, которые в ней используются); эти две проблемы также остаются комплексными и каждая из них в каждой конкретной теории распадутся еще на несколько проблем.

Комплексная проблема машинного обучения в алгебраических байесовских сетях в настоящее время видится как состоящая из следующих проблем: проблема формирования первичной структуры, проблема модификации первичной структуры с учетом некоторых дополнительных требований, которые накладываются особенностями алгоритмов логико-вероятностного вывода и могут также возникать по другим причинам, проблема машинного обучения вторичной глобальной структуры сети по ее первичной структуре, проблемы машинного обучения других глобальных структур сети по ее первичной структуре, проблема машинного обучения параметров сети — скалярных либо интервальных оценок истинности конъюнктов, причем выбор вида оценок также тесно увязан с машинным обучением. В комплексную проблему машинного обучения алгебраических байесовских сетей также входят проблемы алгоритмизации поиска «канонического представителя» со скалярными оценками для отдельного фрагмента или всей сети, в которых имеются интервальные оценки, проблема модификации уже существующей или строящейся сети для учета вновь поступивших данных или знаний.

Марковские сети

Родственной алгебраическим байесовским сетям вероятностной графической моделью являются марковские сети (рис. 2) [2].

Моделью фрагмента знаний выступают узлы такой сети, причем каждому узлу u_{ij} приписывается конечный набор состояний (обобщения в сторону бесконечного числа возможных состояний и др. в докладе не рассматриваются), каждому состоянию приписывается его потенциал — неотрицательная вещественная функция $\varphi(u_{ij}; \cdot)$. Существенной особенностью потенциала любого состояния узла является то, что он зависит от значения состояния самого узла, а также от значений состояний марковского покрытия узла, под которым [покрытием] в классическом случае понимается набор соседей узла. Марковские сети могут иметь структуру решетки (рис. 2, слева) или

ненаправленного графа (рис. 2, справа); в обоих случаях множество соседей узла состоит из смежных ему узлов, то есть тех узлов, которые непосредственно соединены ребром с рассматриваемым узлом.

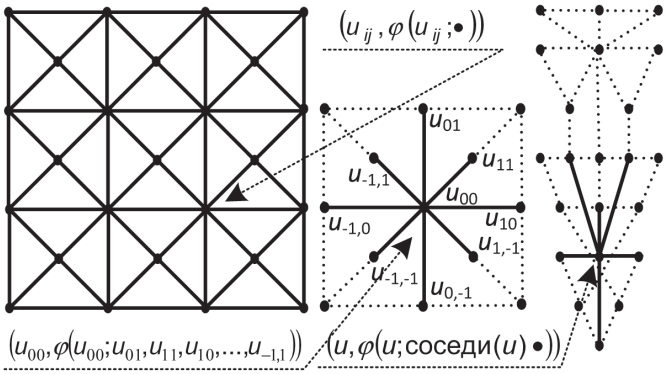


Рис. 2. Марковские сети на двух видах носителей: решетка и граф

Если состояния соседей узла фиксированы, то потенциал состояний самого узла также фиксирован, он не зависит от состояний всех неупомянутых узлов сети — это свойство называется марковским:

$$\varphi(u_{ij}) = \varphi(u_{ij}; \text{соседи}(u_{ij})).$$

Все построения, которые касались марковских сетей в докладе, пока не объясняют, почему марковская сеть рассматривается как *вероятностная* графическая модель. Оценки вероятности возникают тогда, когда потенциал состояния увязывают с вероятностью того, что узел находится в том или ином состоянии. То есть распределение вероятностей состояния узла зависит от величины потенциала каждого состояния узла, при этом очевидно, что такое распределение означиваний фиксировано, если фиксированы состояния соседей узла.

Указанная связь между распределением вероятности состояний узла и состояниями его соседей составляет основу сэмплирования по Гиббсу (более точно было бы говорить о классе методов, объединенных указанным общим названием). Сначала каждому узлу приписывают одно из возможных состояний (в подходах к выбору начальных состояний возможны вариации). Затем, поскольку для каждого узла стало известным состояние его соседей, а значит — и распределение вероятностей его состояний, случайным образом на основе заданного распределения генерируют новое состояние узла (в подходах к обходу узлов, к учету, какое состояние соседей учитывать: строе, новое, применять смешанную стратегию, — также возможны вариации).

Исследование последовательности сложных состояний всей марковской сети (состояние сети — это совокупность состояний всех ее узлов) в динамике представляет интерес для широкого спектра отраслей наук. Например, модель Изинга является марковской сетью, она была разработана в статистической физике для изучения явлений магнетизма. Марковские сети использовались для моделирования процессов распространения слухов в социальных группах с сетевой структурой, и в эпидемиологии — как попытка моделировать распространение инфекций.

Марковские сети являются достаточно гибким аппаратом; потенциалы узлов этих сетей могут быть дополнительно параметризованы, чтобы учесть какие-то особенности «внешней среды». Например, в моделях Изинга в качестве параметров потенциалов могут выступать температура внешней среды и напряженность внешнего магнитного поля. В моделировании социо-инженерных атак параметры могут задавать «жесткость» внешнего контроля действий пользователя; такой параметр может отвечать за скорость налаживания социальных связей злоумышленника и санкционированного пользователя информационной системы.

Байесовские сети доверия

Еще одним классом вероятностных графических моделей, рассматриваемых в настоящем докладе, являются байесовские сети доверия (БСД, рис. 3) [3, 14, 15]. Они кратко определяются как ациклический направленный граф с тензорами условных вероятностей в узлах. Такое определение все же требует дополнительных пояснений.

Во-первых, каждый узел графа, фактически, является, с одной стороны, еще одним вариантом модели фрагмента знаний, а, с другой стороны, случайным элементом. Причем в теории байесовских сетей доверия распределение случайного элемента обусловлено значениями, которые приобретают случайные элементы, сопоставленные узлам-родителям. Это иллюстрируется структурой тензоров условных вероятностей, представленных на рис. 3, а.

Во-вторых, структура байесовской сети допускает ненаправленные циклы (рис. 3, б). Однако при наличии таких циклов становится невозможно пользоваться алгоритмом первичной пропагации и пропагации свидетельств, основанном на распространении сообщений между узлами. В этом случае байесовскую сеть доверия преобразуют особым образом в дерево сочленений, для которого разработаны собственные алгоритмы вероятностного вывода. Отметим, что дерево сочленений служит важным связующим звеном между теориями алгебраических байесовских сетей и байесовских сетей доверия, в частности, в компаративном анализе их вероятностной семантики [15–17].

В-третьих, структура байесовской сети доверия не допускает направленных циклов (рис. 3, в). В западной литературе такие циклы просто ис-

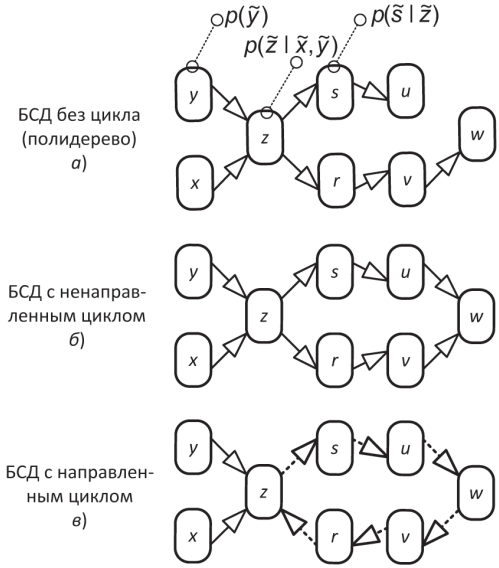


Рис. 3. Байесовские сети доверия с различными структурами графаносителя

ключены из рассмотрения без обоснований такого ограничения. При более подробном рассмотрении вопроса выяснилось, что наличие циклов в байесовской сети доверия моет изменить ее вероятностную семантику так, что БСД-исчисление не сможет быть обобщено на случай направленного цикла. Дело в том, что направленные циклы вместо единственного распределения вероятностей в общем случае задают семейство распределений вероятностей, которое, к тому же, может оказаться пустым, что соответствует случаю противоречивых исходных данных [13]. Теория БСД не располагает инструментами обработки семейств распределений вероятности и не располагает средствами обнаружения противоречий в исходных данных.

Байесовская сеть доверия с допустимой структурой задает единственное распределение вероятностей:

$$p(\tilde{r}, \tilde{s}, \ldots, \tilde{z}) = \prod_{t \in \{r, s, \ldots, z\}} \Pi(\tilde{t}; \text{родители}(t)),$$

причем в левой и правой части равенства используются согласованные означивания соответствующих случайных элементов. Такая вероятностная семантика задает на множестве случайных элементов-узлов достаточно сложную систему условных независимостей, которую в теории байесовских сетей доверия принято именовать d-разделимостью.

Области приложений

Отметим, что конкретные примеры актуальных приложений вероятностных графических моделей рассмотрены в секционных докладах конференции СПИСОК-2013 [7]: Азаров А.А. «Модели комплекса „Информационная система — Персонал — Критические документы“ при оценке защищенности пользователей информационных систем», Суворова А.В. «Грануляция неточных данных в синтезе байесовской сети доверия при исследовании частично наблюдаемых реализаций пуассоновского процесса», Фильченков А.А. «Синтез вторичной структуры алгебраической байесовской сети: компромисс между сложностью и ацикличностью» и др.

Байесовские сети доверия используются в задачах анализа рисков, в том числе эпидемиологического и экономического характера [4, 12]. С их помощью удалось развить подход к косвенной оценке интенсивности рискованного поведения по неполным гранулярным данным [4, 12]. Они (и их обобщения) применяются в распознавании музыкальных произведений и результатов спектрометрии [5, 18], планировании системы тестов оборудования [6].

Марковские сети, помимо традиционных естественнонаучных отраслей своих приложений, нашли применение в области биоинформатики как скрытые марковские модели (СММ; отметим, что последние представимы в виде особых АБС и БСД), а также используются при моделировании объектов социального характера, имеющих структуру сети, отношения, между смежными элементами которой представляются хоть и недетерминированными, но хорошо изученными.

Алгебраические байесовские сети среди трех затронутых в докладе вероятностных моделей имеют самую «молодую» теорию. Хотя она еще развивается, ее применение позволило эксплицировать проблемы, связанные с направленными циклами в байесовских сетях доверия. Кроме того, ведутся работы по представлению скрытых марковских моделей в виде алгебраических байесовских сетей, что, как ожидается, откроет возможность систематически использовать интервальные оценки вероятностей при обучении и использовании СММ. Наконец, предполагается, что АБС будут особенно удобны в машинном обучении при агрегировании данных и знаний с неопределенностью.

Литература

1. Judea Pearl — A.M. Turing Award Winner // ACM. URL: http://amturing.acm.org/award_winners/pearl_2658896.cfm .
2. *Kindermann R., Snell J.L.* Markov random fields and their applications. Am. Math. Soc., 1980. 142 p.
3. *Pearl J.* Probabilistic reasoning in intelligent systems. NY: Morg. Kauf. Publ., 1991. 552 p.

4. *Tulupyev A., Suvorova A., Sousa J., Zelterman D*. Beta prime regression with application to risky behavior frequency screening. Statistics in. Medecine. 2013. No. 32. P. 4044–4056. doi: 10.1002/sim.5820
5. Б*алтийский И.А., Николенко С.И.* Графическая вероятностная модель для оценки схожести гармонии музыкальных произведений // Труды СПИИРАН. 2011. Вып. 18. С. 136–163.
6. *Дорожко И.В., Осипов Н.А.* Методика синтеза оптимальных стратегий диагностирования автоматизированных систем управления сложными техническими объектами с использованием априорной информации // Труды СПИИРАН. 2012. Вып. 20. С. 165–185.
7. Конференция СПИСОК-2013. Секция «Вероятностные графические модели, нечеткие системы и мягкие вычисления» [Электронный ресурс] // Официальный сайт конференции СПИСОК-2013. http://spisok.math.spbu.ru/2013/s20%27.asp .
8. *Мусина В.Ф*. Байесовские сети доверия как вероятностная графическая модель для оценки медицинских рисков // Труды СПИИРАН. 2013. Вып. 24. С. 135–151.
9. *Мусина В.Ф*. Байесовские сети доверия как вероятностная графическая модель для оценки экономических рисков // Труды СПИИРАН. 2013. Вып. 25. С. 235–254
10. *Сироткин А.В*. Вычислительная сложность алгоритмов локального апостериорного вывода в алгебраических байесовских сетях // Труды СПИИРАН. 2011. Вып. 18. С. 188–214.
11. *Сироткин А.В*. Байесовские сети доверия: дерево сочленений и его вероятностная семантика // Труды СПИИРАН. 2006. Вып. 3. Т. 1. С. 228–239.
12. *Суворова А.В., Тулупьева Т.В., Тулупьев А.Л., Сироткин А.В., Пащенко А.Е*. Вероятностные графические модели социально-значимого поведения индивида, учитывающие неполноту информации // Труды СПИИРАН. 2012. Вып. 22. С. 101–112.
13. *Тулупьев А.Л.* Байесовские сети: логико-вероятностный вывод в циклах. СПб.: Изд-во С.-Петерб. ун-та, 2008. 140 с.
14. *Тулупьев А.Л., Николенко С.И., Сироткин А.В*. Байесовские сети: логико-вероятностный подход. СПб.: Наука, 2006. 607 с.
15. *Тулупьев А.Л., Сироткин А.В., Николенко С.И*. Байесовские сети доверия: логико-вероятностный вывод в ациклических направленных графах. СПб.: Изд-во С.-Петерб. ун-та, 2009. 400 с.
16. *Фильченков А.А.* Иерархия глобальных структур алгебраической байесовской сети как система графов и гипер-графов // Научно-техн. вестн. информ-х технол-й, механики и оптики. 2013. Вып. 1. С. 75–81.
17. *Фильченков А.А., Тулупьев А.Л*. Совпадение множеств минимальных и нередуцируемых графов смежности над первичной структурой алгебраической байесовской сети // Вест. СПбГУ. Сер. 1. 2012. Вып. 2. С. 69–78.
18. *Чернявский И.И., Александров А., Николенко С.И.* Метод сегментации результатов MALDI-спектрометрии на основе графических вероятностных моделей // Труды СПИИРАН. 2012. Вып. 21. С. 120–142.

ОПТИМИЗАЦИЯ ИСПОЛЬЗОВАНИЯ КЭШ-ПАМЯТИ В ВЫЧИСЛИТЕЛЬНЫХ ПРОГРАММАХ И ОПТИМИЗИРУЮЩЕЙ КОМПИЛЯЦИИ

Штейнберг Б.Я.

д.т.н.. зав. каф. АДМ мехмата, Южный федеральный университет

Работа поддержана ОАО «Ангстрем»

Введение. Изменилось понятие сложности алгоритма, как модели времени выполнения программы.

В течение нескольких десятилетий с каждым годом уменьшается отношение времени выполнения арифметической операции ко времени считывания аргументов этой операции из памяти [1], [2]. Пятьдесят лет назад умножение вещественных чисел было очень дорогостоящей операцией, а обращения к памяти требовали незначительное время по сравнению с умножением. На сегодняшний день — наоборот, обращение к памяти более, чем на порядок дольше, чем выполнение арифметических операций. Это привело к необходимости использования модулей вспомогательной памяти (кэш или локальная память) с быстрым доступом, но малым объемом.

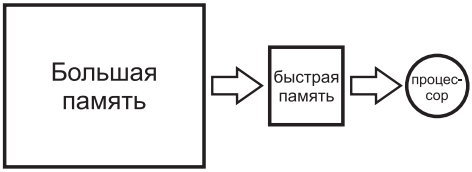


Рис. 1. Иерархия памяти требует корректировки алгоритмов для достижения максимального быстродействия

В массовых процессорах в качестве большой памяти можно рассматривать оперативную память, а в качестве быстрой памяти может рассматриваться кэш память. В процессорах IBM Cell быстрой памятью служит локальная (программируемая, в отличие от кэш) память каждого процессорного элемента [8]. Данная модель может описывать и задачи, в которых большая память — это внешняя память, а быстрая — это оперативная.

Сочетание иерархии памяти и параллельности

Разработка быстрых программ должна быть ориентирована на архитектуру вычислительной системы. В [3] показано, что требования к соотношению объема быстрой памяти и быстродействию процессора (количество

вычислительных ядер) определяются задачами, которые предполагается на этом компьютере решать: для одних задач оптимальной будет одна архитектура, а для других — другая.

Современное поколение вычислительных систем, несмотря на их многообразие, характерно тем, что время передачи данных с кристалла оперативной памяти на кристалл процессора во много раз больше времени обработки этих данных. Многоядерность процессора имеет эффект для тех задач, в которых с малым числом данных выполняется много операций.

Площадь на кристалле процессора распределяется между локальной быстрой памятью (кэш) и вычислительными ядрами. И увеличение объема кэш памяти и увеличение количества вычислительных ядер могут повышать быстродействие процессора. Чем меньшую площадь кристалла будут занимать вычислительные ядра, тем больше может быть кэш память. Уже появляются многоядерные процессоры с количеством ядер порядка 100, но такое увеличение приводит к уменьшению памяти на этом же кристалле — ускорятся ли при этом вычисления? Оказывается, что для разных алгоритмов по-разному. Алгоритмы можно разделить на несколько типов:

1. алгоритмы, для которых увеличение кэш памяти и увеличение количества ядер не приводят к увеличению быстродействия – время вычисления сводится ко времени перекачки данных с кристалла памяти на кристалл процессора,
2. алгоритмы, для которых существенно увеличение кэш памяти,
3. алгоритмы, для которых существенно увеличение количества ядер,
4. алгоритмы, быстродействие которых зависит от баланса между объемом кэш памяти и количеством вычислительных ядер (перемножение матриц).

Примером алгоритмов первого типа является скалярное произведение векторов. К алгоритмам второго типа относится сортировка большого массива. К алгоритмам третьего типа можно отнести вычисление степенных рядов с аналитически вычисляемыми коэффициентами (например, для вычисления ряда Тейлора exp(x) следует из большой памяти считать только переменную x, а сам ряд можно вычислять частями параллельно). К алгоритмам четвертого типа относится перемножение матриц.

Если процессор имеет программируемую архитектуру и допускает только изменение связей между вычислительными устройствами на кристалле, то его эффективность все равно не выйдет за пределы класса алгоритмов, который определяется соотношением площади кристалла, отведенной для памяти, к площади кристалла, отведенной для вычислительных ядер.

Разбиение задач на подзадачи

Иерархия памяти предполагает разбиение задачи на иерархию подзадач: на каждом уровне памяти должна решается подзадача соответствующего уровня.

Адекватное разбиение задачи на иерархию подзадач определяет эффективность вычислений. Следует подчеркнуть, что компиляторы делают такое разбиение автоматически, и, как правило, без оптимизации. Рассмотрим программу перемножения матриц.

```
do i = 1, n
do j = 1, n
do k = 1, n
X(i,j) = X(i,j)+A(i,k)*B(k,j).
```

Если матрица не помещается в кэш память, то в кэш попадут строка матрицы A и столбец B, которые необходимы для вычисления самого вложенного цикла. В этом случае компилятор разбивает задачу перемножения матриц на подзадачи, состоящие в вычислении скалярных произведений. Тогда на каждую арифметическую операцию будет приходиться одно обращение к памяти (чтение). Если чтение чисел из оперативной памяти в 10 раз дольше перемножения чисел, то процессор будет загружен лишь на 1/10 своей мощности. Для повышения эффективности вычислений в этой задаче переходят к блочному произведению матриц.

Блочное перемножение матриц выглядит примерно следующим образом.

```
f = n/m
do i1 = 1, f
do j1 = 1, f
do k1 = 1, f
  do i = 1, m
  do j = 1, m
  do k = 1, m
XX(i1,j1)(i,j) = XX(i1,j1)(i,j) + AA(i1,k1)(i,k) * BB(k1,j1)(k,j)
```

Блоки подбираются так, чтобы 3 блока поместились в быстрой памяти

A_{11}	A_{12}	A_{13}	A_{14}
A_{21}	A_{22}	A_{23}	A_{24}
A_{31}	A_{32}	A_{33}	A_{34}
A_{41}	A_{42}	A_{43}	A_{44}

Рис. 2. Разбиение матрицы на блоки

Основная цель разбиения задач на подзадачи состоит в том, чтобы данные каждой подзадачи помещались в быстрой памяти вычислительного устройства и чтобы подзадач было мало. Оба условия противоречат друг другу. Баланс между этими условиями приводит к необходимому условию разбиения задачи на подзадачи:

подзадачи должны формироваться так, чтобы с данными в ограниченной объемом быстрой памяти можно было выполнять много операций;

Если в вычислительной архитектуре используется кэш память нескольких уровней, то для задачи перемножения матриц перемножение блоков следует опять разбить на блоки. Получается иерархия подзадач, соответствующая иерархии памяти. Следует отметить, что в рекордных по быстродействию алгоритмах перемножения матриц блоки могут быть не квадратными [4].

Описание данных для быстрых вычислений

Не всякая задача может быть разбита на подзадачи, подобные исходной с меньшим объемом данных, как это происходит с перемножением матриц. Например, задача сортировки большого массива.

Для эффективной сортировки большого массива сортируемый массив должен быть разбит на части, которые могут полностью попасть в быструю память. Каждая такая часть может быть эффективно отсортирована в быстрой памяти. Затем требуется провести слияние отсортированных массивов. Попарные слияния массивов не продуктивны, поскольку на каждую операцию сравнения приходится два обращения к большой памяти. В этом случае выгодной оказывается множественное слияние. Множественное слияние предполагает, что в быструю память попадут минимальные элементы сразу нескольких сливаемых массивов. Пусть количество сливаемых массивов равно k. Элементы сливаемых массивов должны быть организованы в специальную структуру данных, которая за $O(\log_2(k))$ шагов позволяет выполнить следующие операции:

1. найти минимальный $X_{\min}$ из этих k элементов и сбросить его в большую память в массив, который получается в результате слияния,
2. считать из большой памяти в быструю очередной элемент из того из сливаемых исходных массивов, к которому принадлежал $X_{\min}$ и разместить его в искомой структуре.

Искомой структурой может служить, например, 2–3 дерево. [5]

Компиляторы имеют стандарты распределения данных. Например, компилятор языка ФОРТРАН размещает матрицы в оперативной памяти по столбцам, а компиляторы языков Си и Паскаль — по строкам. Для минимизации кэш промахов могут использоваться нестандартные размещения многомерных массивов. Например, блочное размещение позволяет получить произведение матриц быстрее, чем у программ известных библиотек [7].

Для распределенной памяти в группе ОРС [6] исследуются размещения с перекрытиями, суть которых в том, чтобы потратив (до 7%) память на дублирование некоторых данных, выиграть в быстродействии (несколько десятков процентов). Ведутся работы по автоматизации на основе Оптимизирующей распараллеливающей системы [6] блочных распределений массивов в оперативной памяти и блочно-аффинных размещений массивов с перекрытиями в распределенной памяти.

Следует отметить, что рекордная производительность достигается непросто, например, рекордная программа перемножения матриц занимает порядка 1000 строк кода, среди которых много ассемблерных, учитывающих различные особенности архитектуры процессора. Это примерно в 100 раз больше объема кода обычной программы. Программы с оптимизированным размещением данных в распределенной памяти также занимают существенно больший объем кода, чем обычные параллельные программы. Кроме увеличенного времени разработки рекордные программы требуют очень высокой квалификации программистов. Сокращение времени разработки может быть достигнуто средствами автоматической оптимизации, которые, на сегодняшний день, развиты недостаточно.

Сочетание использования параллелизма и иерархии памяти рассматривалось для задач парного выравнивания последовательностей (в биоинформатике) в работе [9].

Литература

1. *Корнеев В.В.* Проблемы программирования суперкомпьютеров на базе многоядерных мультитредовых кристаллов // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: Труды Всероссийской суперкомпьютерной конференции (21-26 сентября 2009 г., г. Новороссийск). — М.: Изд-во МГУ, 2009.
2. *Эйсымонт Л.К., Горбунов В.С.* На пути к экзафлопсному суперкомпьютеру: результаты, направления, тенденции. МСКФ 2013, М., 1 ноября 2013. http://www.osp.ru/docs/mscf/mscf-001.pdf (дата обращения 18.07.2013)
3. *Штейнберг Б.Я.* Зависимость оптимального распределения площади кристалла процессора между памятью и вычислительными ядрами от алгоритма. РАСО´2012/ Труды международной конференции «Параллельные вычисления и задачи управления». М., 24–26 октября 2012 г., ИПУ РАН, с. 99–108.
4. *Kazushige Goto, Robert A. van de Geijn*. Anatomy of High-Performance Matrix Multiplication. ACM Trans. Math. Softw., Vol. 34, No. 3. (May 2008), pp. 1–25.
5. *А.Ахо, Дж. Хопкрофт, Дж. Ульман.* Построение и анализ вычислительных алгоритмов. М., Мир, 1979, 526 с.
6. Распараллеливающая система OPS — URL:http://www.ops.rsu.ru/ Дата обращения: 05.09.2013
7. Быстрое перемножение матриц на сайте оптимизирующей распараллеливающей системы http://ops.opsgroup.ru/downloads/dgemm.zip.
8. *Линев А.В., Боголюбов Д.К., Бастраков С.И.* Технологии параллельного программирования для процессоров новых архитектур. Учебник / Под ред. В.П. Гергеля.

М.: Изд-во Московского ун-та, 2010, 160 с. — (Серия «Суперкомпьютерное образование»)

9. *Абу-Халил Ж.М., Морылев Р.И., Штейнберг Б.Я.* Параллельный алгоритм глобального выравнивания с оптимальным использованием памяти // Современные проблемы науки и образования. — 2013. — №1; (Электронный журнал http://www.science-education.ru/107-8139).

ИССЛЕДОВАНИЯ И ПРЕПОДАВАНИЕ В ОБЛАСТИ СОВРЕМЕННЫХ ТЕХНОЛОГИЙ И ИНСТРУМЕНТОВ ПРОГРАММИРОВАНИЯ И ОБЛАЧНЫХ ВЫЧИСЛЕНИЙ

Сафонов В.О.

д.т.н., профессор кафедры информатики СПбГУ, vosafonov@gmail.com

Аннотация. Рассмотрены уникальный опыт и результаты автора и его научной школы в области современных технологий и инструментов программирования: аспектно-ориентированного программирования (АОП), Java-технологии, облачных вычислений на новейшей платформе Microsoft Windows Azure, управления знаниями, параллельного программирования. Описаны опыт и многочисленные новые публикации автора по АОП, облачным вычислениям, операционным системам, параметризованным типам данных, Java-технологии, его Интернет-курсы в этих областях, используемые автором и его школой для преподавания на мат-мехе.

Современные технологии и инструменты разработки ПО

Современное программное обеспечение, технологии его разработки и использования развиваются быстрыми темпами. В связи с этим требуется и корректировка исследований, и постоянное изменение учебных программ университетского образования. Вот перечень лишь некоторых новых направлений в области развития ПО:

- *Облачные вычисления* — технологии разработки и использования масштабируемого набора виртуализованных ресурсов — программ и данных, доступных клиентам через Web-интерфейс браузеров и реализованных на компьютерах центров обработки данных поставщиков облака. Наиболее популярная платформа облачных вычислений — Microsoft Windows Azure (http://windows.azure.com)
- *Сервисно-ориентированные архитектуры* — разработка и использование ПО в виде расширяемого набора слабо связанных, разработанных по единому стандарту Web-сервисов
- *Надежные и безопасные вычисления (Trustworthy computing)* — разработка программного обеспечения, удовлетворяющего принципам безопасности, надежности и защиты конфиденциальной информации
- *Аспектно-ориентированное программирование* — современный подход к программированию с целью модуляризации сквозной функциональности в виде особых модулей — аспектов, которые используются путем автоматической вставки (weaving) их фрагментов в целевые приложения

- *Семантические вычисления (Semantic computing)* — вычисления с использованием знаний в виде онтологий, наборов правил, фреймов, нечетких знаний и других видов знаний
- *Мобильные вычисления* — разработка и использование ПО для мобильных устройств
- *Использование формальных спецификаций и верификации* программ для повышения их надежности и безопасности
- *Параллельное программирование* — развитие методов и инструментов поддержки возможностей современных многоядерных и многопроцессорных компьютеров для параллельных вычислений.

Современные инструменты разработки программ и обучения

Вот перечень некоторых современных технологий и инструментов, развитие и изучение которых весьма важно.

- .NET 4.5, Visual Studio 2012, & Team Foundation Server (Microsoft). Платформа .NET — многоязыковая платформа надежного и безопасного объектно-ориентированного программирования.
- SLAM, Z3, Pex, Boogie — инструменты верификации программ и генерации тестов (Microsoft Research).
- Windows Azure (Microsoft) — новейшая платформа облачных вычислений;
- Hawaii — проект по разработки облачных приложений в системе Azure для мобильных устройств;
- Java 7, NetBeans (Oracle) — новые инструменты Java-технологии;
- Aspect.NET (www.aspectdotnet.org) — инструментарий АОП для платформы .NET, разработанный нашей группой и используемый в 26 странах мира;
- Knowledge.NET (www.knowledge-net.ru) — инструментарий инженерии знаний для платформы .NET, обеспечивающий расширение языка C# средствами представления знаний для разработки интеллектуальных программных решения.

Исследования и результаты

Вот некоторые направления и результаты наших исследований в указанных областях.

- Система Aspect.NET [4, 7, 15]; ее версия для академической версии .NET (AspectRotor).
- Библиотеки аспектов для Web-программирования, безопасности, надежных и безопасных вычислений, контрактного проектирования (проблемно-ориентированное АОП — domain-specific AOP);

- Применение АОП для облачных вычислений в системе Azure (интеграция аспектов Aspect.NET в облачные приложения)
- Система Knowledge.NET; интеллектуальные решения на ее базе (дипломные работы)
- Предложения по расширению Java generics параметрами-константами и их экспериментальная реализация в рамках проекта OpenJDK [1].

Опыт исследований и преподавания в сфере облачных вычислений

Научная школа автора добилась уникальных результатов, одной из первых в России начав преподавание облачных вычислений и исследования в данной области. Вот хронология этих результатов:

- 2011: Грант Microsoft Research (один из первых в Европе) — академический доступ к Azure для В.О. Сафонова и его 50 учеников на 180 дней
- Декабрь 2011: Публикация учебного курса «Платформа облачных вычислений Microsoft Windows Azure» (ИНТУИТ.РУ) [13]
- 2011–2012: Его использование в лекциях по курсу «Архитектуры и модели программ и знаний» (4 курс) [14]
- 2011–2013: Исследования по применению АОП и системы Aspect.NET на платформе Azure (В.О. Сафонов, Д.А. Григорьев, А.В. Григорьева, Нгуен Ван Доан)
- 2012, 2013: Новые гранты MSR — академический доступ к Azure (2012 — 180 дней; 2013 — 1 год)
- 2012: Использование материалов по Azure в курсе «Компьютерные сети» (2 курс) — теория и практика облачных проектов
- 2012: Публикация книги «Платформа облачных вычислений Microsoft Windows Azure» [1]. Эта книга — одна из первых научно-педагогических публикация по облачным вычислениям в России.
- 2013: Разработка нового учебного курса «Развитие платформы облачных вычислений Microsoft Windows Azure» (в нем будет описан новый интерфейс Azure). Планируется публикация курса на сайте ИНТУИТ.РУ и выпуск книги по этому курсу.

Преподавание современных технологий и инструментов разработки ПО автором и его школой на мат-мехе СПбГУ

Автором курса преподаются на мат-мехе СПбГУ следующие дисциплины, неизменно вызывающие большой интерес студентов. Программы этих дисциплин постоянно обновляются автором:

- *Компьютерные сети*, обязательный курс, 30 часов лекции + 30 часов практика, (2 курс, фундаментальная информатика и информационные

технологии, с 2012 г.); курс включает большой материал по облачным вычислениям и платформе Microsoft Windows Azure
- *Java-технология*, спецкурс, 1 год (3 курс) [5, 6]
- *Программирование для Microsoft.NET*, спецсеминар, 1 семестр (4 курс)
- *Архитектуры и модели программ и знаний*, обязательный курс по специализации, 64 часа (4 курс)
- *Компиляторы*, обязательный курс по специализации, 64 часа (5 курс) [8, 17].

Опыт преподавания облачных вычислений

Ныне, благодаря деятельности автора и его школы, облачные вычисления вошли в практику обучения ИТ-студентов мат-меха. Они вызывают огромный интерес студентов и аспирантов.

При преподавании приходится учитывать и излагать специфику облачных вычислений, например: единица хранения информации — Web-сайт; облако реализовано в виде набора облачных сервисов; единицами вычислений в облаке являются Web-роли (Web-приложения) и worker-роли (независимые фоновые процессы); возможно использование через облако виртуальных машин.

Помимо преподавания теории облачных вычислений, мы организовали практику студентов в Windows Azure: Учебные проекты по сетям для 2 курса выполняются под руководством аспиранта В.А. Васильева.

Публикации по технологиям разработки программ и облачным вычислениям

В списке литературы приведен, без преувеличения, уникальный перечень публикаций автора по современным технологиям разработки программ, включая облачные вычисления [1–17]. Это монографии, учебники, статьи и Интернет-курсы, написанные автором. Достаточно сказать, что за 2002–2013 г. автором написано 8 книг (всего автор на данный момент написал 15 книг), опубликовано на академических сайтах более 20 Интернет-курсов, опубликовано около 30 статей по преподаванию и исследованиям в области технологий и инструментов разработки программ. Все эти материалы могут быть использованы другими преподавателями и, конечно, студентами и аспирантами.

Литература

1. *Сафонов В.О.* Параметризованные типы данных: история, теория, реализация и применение. — СПб.: изд-во СПбГУ, 2013, 130 с. (в печати).
2. *Сафонов В.О.* Платформа облачных вычислений Microsoft Windows Azure. — М.: ИНТУИТ.РУ. БИНОМ. Лаборатория знаний, 2012, 235 с.

3. *Сафонов В.О.* Основы современных операционных систем. — М.: ИНТУИТ.РУ. БИНОМ. Лаборатория знаний, 2011, 583 с.
4. *Сафонов В.О.* Аспектно-ориентированное программирование. СПб.: изд-во СПбГУ, 2011, 104 с.
5. *Сафонов В.О.* Введение в Java-технологию. Lambert Academic Publishing, Saarbrucken, Germany, 2011, 232 pp.
6. *Сафонов В.О.* Введение в Java-технологию. — СПб: Наука, 2002, 187 с.
7. *Safonov V.O.* Using aspect-oriented programming for trustworthy software development. — Wiley, 2008
8. *Safonov V.O.* Trustworthy compilers. — Wiley, 2010
9. *Сафонов В.О.* Серия статей по новым технологиям разработки ПО и их преподаванию, «Компьютерные инструменты в образовании», 2003–2012, около 30 статей. Доступны в Интернете
10. *Сафонов В.О.* Экспертные системы — интеллектуальные помощники специалистов. — СПб.: изд-во «Знание», 1992
11. *Сафонов В.О.* Языки и методы программирования в системе «Эльбрус». — М.: Наука, 1989, 390 с. (Глава 5. ТИП-технология программирования)
12. *Сафонов В.О.* Основы современных операционных систем. Учебный курс (на сайтах ИНТУИТ.РУ и Microsoft Rus). Второе место в мире по числу скачиваний (Microsoft Faculty Connection); рейтинг 4.64; около 2600 студентов онлайн (ИНТУИТ.РУ). http://www.intuit.ru/studies/courses/641/497/info
13. *Сафонов В.О.* Платформа облачных вычислений Microsoft Windows Azure. http://www.intuit.ru/studies/courses/2314/614/info. Публикация: декабрь 2011 г. Около 550 онлайн-студентов (ИНТУИТ.РУ)
14. *Сафонов В.О.* Архитектуры и модели программ и знаний. Материалы курса. Декабрь 2009. https://www.facultyresourcecenter.com/curriculum/pfv.aspx?ID=8480&Login=
15. Система аспектно-ориентированного программирования Aspect.NET https://www.facultyresourcecenter.com/curriculum/pfv.aspx?ID=8234&Login=
16. *Сафонов В.О.* Инженерия надежных и безопасных программ. Учебный курс для студентов 4 курса. Ноябрь 2007 г. https://www.facultyresourcecenter.com/curriculum/pfv.aspx?ID=8143&Login=
17. *Safonov V.O.* Trustworthy Compiler Development. Graduate university course. http://www.facultyresourcecenter.com/curriculum/pfv.aspx?ID=7698. February 2010

АЛГОРИТМИЧЕСКАЯ СЛОЖНОСТЬ РАЗРЕШИМОСТИ ЯЗЫКА СВОЙСТВ ПРОГРАММ-ФУНКЦИЙ НА КОМПЬЮТЕРЕ

Н.К. Косовский

Аннотация. Для ряда языков программирования, в частности, версий языков Паскаль и Турбо/Вижуал Пролог, вводится математическое понятие программы-функции, реализуемой на компьютере, как сужение класса алгоритмов **FP** для более адекватного отражения конечной памяти *m Kbyte* компьютера, на котором реализуется программа любого из рассматриваемых языков программирования. Изучена сложность распознавания логико-математических свойств таких программ-функций, рассматриваемых как «чёрные ящики». Вводится язык логики предикатов для формулировки свойств такого рода программ, являющихся частичными функциями. Доказывается **P-SPACE**-полнота каждой элементарной теории, построенной в конечной сигнатуре из такого рода программ-функций на основе предикатов вычислимости, условного равенства и условных неравенств. Тем самым устанавливается возможность использования языка первого порядка для описания логико-математических свойств программ-функций и программ-предикатов путём доопределения до всюду определённых используемых программ-функций. Для доопределения используется специальное дополнительное значение «?». Программы-предикаты доопределяются до всюду определённых константой *false*. Это позволяет во многих случаях заменить традиционно используемые математические понятия алгоритмов предлагаемыми программами-функциями в рамках математической информатики.

Ключевые слова: языки программирования, математические понятия алгоритмов, классы алгоритмов **FP**, **FP-SPACE**, программы, реализуемые на компьютере.

Введение

В 30-х годах прошлого века было введено математическое понятие алгоритма. Вплоть до 70-х годов прошлого века основные усилия математиков, работающих с этим понятием, были сконцентрированы на проблемах существования (а точнее, несуществования) алгоритмов для решения самых разнообразных задач.

Благодаря теореме Кука, доказанной в 1971 году, многие математики получили возможность использовать её для доказательства NP-полноты многих тысяч разнообразных задач дискретной математики. Доказательство NP-полноты той или иной задачи по существу означает, что в ближайшее время, например, в течение двух лет, не будет разработан алгоритм, решающий эту задачу за время, полиномиальное относительно длины записи исходных

данных. Дело в том, что доказательство существования такого алгоритма влечёт положительное решение проблемы **P=NP**, объявленной одной из семи труднейших задач математики. Полиномиальный алгоритм в достаточной мере приспособлен для массового использования на компьютере. В то же время в теории реализуемых массовых компьютерных вычислений заслуживают внимания и более сложные алгоритмы, например, NP-трудные, P-SPACE-полные, не имеющие в настоящее время полиномиального алгоритма для их решения. В ближайшем будущем вряд ли возможно изменение такой ситуации.

Ниже получены результаты для конечных сигнатур логики первого порядка с функциями и предикатами, реализуемыми на любом языке программирования версий языка Паскаль или Турбо/Вижуал Пролог при использовании типов данных конечного объёма. При этом возможны два варианта использования таких типов данных. Во-первых, явное указание числа различных вариантов этих типов данных в использованном языке программирования. Во-вторых, явное ограничение сверху на объём памяти, используемой для вычисления на компьютере.

Широкую известность получили машины Тьюринга, алгоритмы Маркова, РАМ и РАСП. Много позже были введены и многие другие математические понятия алгоритма, например, предложенное автором понятие алгоритма Маркова-Поста [7]. В последней трети XX века созрело убеждение, что класс эффективных алгоритмов совпадает с классом FP. Здесь и далее FP — класс функций, вычислимых за полиномиально ограниченное число шагов машины Тьюринга, измеряемое относительно длины записи значений аргументов.

Приведу формулировку обобщённого тезиса Чёрча из [8]: «A function computable in polynomial time in any *reasonable* computational model using a *reasonable* time complexity measure is computable by a deterministic Turing machine in polynomial time», т. е. «Функция, вычислимая за полиномиальное число шагов в любой *разумной* вычислительной модели, использующей *разумное* измерение временной сложности, вычислима на детерминированной машине Тьюринга не более, чем за полином шагов.

Математическая модель RAM+BOOL (т. е. RAM [1] с дополнительными поразрядными логическими операциями) кажется достаточно подходящим математическим понятием алгоритма для того, чтобы более адекватно (относительно операций компьютеров) отразить число шагов вычисления, чем понятие машины Тьюринга. Это связано с тем, что в RAM имеется, например, косвенная адресация, широко используемая при обработке массивов в компьютере. Однако за *n* шагов на RAM+BOOL можно вычислить, например, 2^(2^n), что гораздо менее адекватно числу шагов вычисления этой функции на компьютере.

Более того, из работы [5] следует, что на RAM+BOOL за полиномиальное число шагов (при равномерном весе команд [1]) можно решить задачу ВЫПОЛНИМОСТЬ, которая является NP-полной задачей [4].

Для обеспечения принадлежности классу FP (т. е. классу алгоритмов, вычислимых за полиномиальное время на машине Тьюринга) предлагалось использовать введенное в [5] понятие объёма изменений, представляющее собой произведение числа шагов вычисления на длину используемой памяти.

Утверждение [5]. *Класс всех программ, написанных для RAM+BOOL, объём изменений которых ограничен сверху полиномом от длины записи исходных данных, совпадает с классом* **FP**.

Применение компьютера для реализации программы на любом языке программирования предполагает использование памяти, явно ограниченной сверху константой.

Исходная версия языка Паскаль, предложенная Н. Виртом, имеет только один тип данных бесконечного объёма (*size*), а именно тип *file*. В более поздних версиях языка Паскаль используются динамические массивы, также являющиеся типом данных бесконечного объёма. На языке Турбо/Визуал Пролог в качестве типов данных бесконечного объёма используются файлы и списки элементов (одного и того же типа), а также термы только с неинтерпретируемыми именами функций. Понятие строки символов, не ограниченной заранее длины, также является примером типа данных неограниченного объёма.

Следует отметить, что в дискретной математике, иногда, используются бесконечные множества, например, множество всех слов в заранее выбранном алфавите, множество всех целых чисел, множество всех рациональных чисел и т. д. При этом часто не учитывается способ записи этих элементов, например, по какому основанию используется система записи чисел. В рамках математической информатики представляет интерес изучение свойств алгоритмов лишь из класса FP-SPACE, т. е. класса алгоритмов, вычислимых на машине Тьюринга на ленте, длина которой ограничена полиномом от длины исходных данных. Это, в частности, демонстрируют теоремы, доказанные ниже.

Ранее полученные результаты: ПАСКАЛЕВИДНЫЕ ФУНКЦИИ КАК МАТЕМАТИЧЕСКОЕ ПОНЯТИЕ АЛГОРИТМА

Как уже было сказано, полиномиальное число шагов для разных математических понятий алгоритмов не обязательно обеспечивает его принадлежность классу **FP**. В то же время, программирование на машине Тьюринга не вполне адекватно современной практике программирования. В связи с этим введем математическое понятие целочисленного алгоритма на основе широко распространенного языка паскаль. Более точно, в осно-

ве рассматриваемого понятия лежит предложенный Н. Виртом язык паскаль [6] с включением динамических массивов (т. е. массивов с динамически определяемыми верхними границами), а не другие его различные реализованные версии.

Пусть в языке паскаль переменные типа *integer* могут принимать в качестве значения любое целое число (т. е. целое число с любой сколь угодно большой длиной записи), а другие основные типы переменных не используются. Как иногда говорят, возможно использование только констант и целых чисел произвольной разрядности.

Определение. *Паскалевидной функцией (целочисленным алгоритмом) назовем функцию предложенного Н. Виртом языка паскаль, обрабатывающую ее фактические параметры — целые числа, поступающие на ее вход, и не использующую файлы, записи, множества и указатели, а также процедуры и функции, имеющие в качестве параметров свои процедуры и функции.*

Основой при получении оценок сложности вычислений является определение того, что же считать шагом вычисления и что считать длиной используемой памяти. С определением шага вычисления, как правило, трудностей не возникает. Но уже отмечено, что на разных моделях вычисления функция 2^(2^n) вычисляется за очень разное число шагов.

Определение. *Вычисление каждого оператора языка паскаль из определения функции (включая оператор цикла и оператор присваивания выражения) считается за один шаг. Каждое обращение к ранее определенной функции или процедуре (т. е. ее вызов) также считается за один шаг. К каждому шагу прибавляется число всех дополнительных шагов, необходимых для его выполнения, включая и все рекурсивные вызовы. (Такое обращение возможно и при вычислении арифметического или булева выражения.)*

На каждом шаге вычисления функции будем дополнительно вычислять длину записи вычисления на этом шаге.

Пусть *t* — арифметическое или булево выражение. Процесс его вычисления может быть представлен в виде корневого дерева. Глубиной записи *t* назовем глубину этого дерева.

Определение. *Длиной записи вычисления выражения t на глубине i назовем сумму длин абсолютных величин всех подвыражений глубины i выражения t.*

Длиной записи вычисления выражения t назовем максимум по всем i (от 1 до глубины t) длин записи вычисления t на глубине i.

Определение. *На начальном шаге длина записи вычисления совпадает с длиной записи исходных данных (значений для аргументов функции). На заключительном шаге вычисления функции длина записи вычисления совпадает с длиной записи абсолютной величины результата вычисления вызываемой функции.*

Под длиной записи промежуточных вычислений (т. е. используемой памяти) понимаем максимум длины записей вычисления по всем шагам этого вычисления, включая начальный и заключительный.

Для удобства читателя предполагаем, что все числа записаны в десятичной системе счисления (хоть основание системы счисления, отличное от единицы, совершенно не принципиально).

Определение. *Дважды полиномиальной паскалевидной функцией над конечным списком паскалевидных функций S назовем такую паскалевидную функцию, для которой как ее число шагов, так и ее длина записи промежуточных данных (без учета их внутри вычисления функций из S) не превосходят некоторого полинома от длины записи исходных данных.*

Теорема. *Класс паскалевидных функций (в том числе и класс функций, задающих предикаты), дважды полиномиальных над списком S, все функции из которого принадлежат классу* **FP***, совпадает с классом* **FP** *(соответственно с классом* **P**).

Аналогичные теоремы были доказаны для языков Турбо-Пролог, Визуал-Пролог и Рефал-5 в [9].

Исходные определения

Поскольку язык классического исчисления предикатов предназначен для сигнатур, содержащих только всюду определённые функции и предикаты, то каждый тип данных для его использования можно дополнить неопределённым значением, возникающим при бесконечном зацикливании или отсутствием необходимой памяти на компьютере. Это значение можно обозначать, например, знаком «?». Для программ-функций, реализованных на компьютере, ниже будет доказана теорема о существовании алгоритма, решающего проблему остановки (т. е. применимости программы к данным). Это является существенным аргументом для использования языка классической логики первого порядка для описания логико-математических свойств программ-функций, реализованных на компьютере.

Отметим, что в области значений предметных переменных будет использоваться новая константа, обозначенная посредством «?». Среди констант в сигнатуре знак ? будет присутствовать. Среди элементов носителя он будет также присутствовать. Если значение хоть одного из её аргументов равно «?», то значение всей функции считается равным «?». Программы-функции с результатами типа *boolean* доопределяются константой *false*, т. е. если значение хоть одного из аргументов равено «?», то значением предиката является константа *false*.

Пусть x — кортеж аргументов программы-функции F.

Выражение $!F(x)$ означает отсутствие бесконечного зацикливания или нехватка памяти при вычислении значения $F(x)$, т. е. неверно, что $F(x) = ?$.

Пусть $t1$, $t2$ — термы.

Выражение $t1 \sim = t2$ означает условное равенство левой и правой частей. То есть обе части одновременно не определены или они одновременно определены и результаты их вычисления совпадают. Фактически, это обычное равенство вычисленных значений при использовании знака «?».

Для дальнейшего изложения зафиксируем положительные целые числа *m* и *M*. Предполагается использование компьютера с доступной памятью *m Kbyte* и языка первого порядка в сигнатуре с носителем, совпадающим с множеством всех элементов компьютерного типа *integer*, дополненным знаком ?. Этот тип представляет собой знак ? или остатки от деления на *2^(2^M)* в полусегменте *[–2^(2^M–1), 2^(2^M–1))*.

При $M = 4$ этот полуинтервал совпадает с множеством целых чисел компьютерного типа *integer* IBM-совместимых персональных компьютеров. При $M = 5$ это будут числа типа *double-integer* для таких же компьютеров.

Рассматриваемая сигнатура состоит из

- носителя, включающего знак «?» и все целочисленные константы из *[–2^(2^M–1), 2^(2^M–1))*;
- констант, содержащих знак «?» и все целочисленные константы из *[–2^(2^M-1), 2^(2^M-1))*;
- конечного числа вычисляемых на компьютере с памятью *m Kbyte* программ-функций, аргументы и результаты которых являются знаком «?» или целыми числами из рассматриваемого диапазона;
- конечного числа вычисляемых на компьютере с памятью *m Kbyte* программ-предикатов, аргументы которых являются знаком «?» или целыми числами из рассматриваемого диапазона, а результаты имеют тип *boolean*.

В качестве исходных предикатов используются только следующие предикаты: применимость !, условное равенство $\sim =$ и условные неравенства.

Под предикатами условных неравенств, обозначаемых посредством $\sim <$, $\sim \leqslant$, $\sim >$, $\sim \geqslant$, понимаются всюду определённые предикаты, каждый из которых истинен тогда и только тогда, когда их оба аргумента бесконечно зацикливаются (т. е. равны ?) или оба аргумента применимы и находятся соответственно в отношении $<$, $\leqslant$, $>$, $\geqslant$.

Кроме этого предикатами условного неравенства, обозначаемыми посредством $\subset \sim <$, $\subset \sim \leqslant$, $\supset \sim >$, $\supset \sim \geqslant$, понимаются всюду определённые предикаты, каждый из которых истинен, если оба аргумента определены и находятся соответственно в отношении $<$, $\leqslant$, $>$, $\geqslant$.

Точнее, отношения $\subset \sim <$ и $\subset \sim \leqslant$ истинны тогда и только тогда, когда

– если первый аргумент определён, то и второй аргумент тоже определён,
– и они находятся соответственно в отношении $<$ или $\leqslant$.

Отношения $\supset \sim >$ и $\supset \sim \geqslant$ истинны тогда и только тогда, когда

– если второй аргумент определён, то и первый аргумент тоже определён,

– и они находятся соответственно в отношении $>$ или $\geqslant$.

Логико-математические формулы строятся посредством языка предикатов первого порядка традиционным способом с помощью знака отрицания $\neg$, пропозициональных логических связок и кванторов по целым числам из полусегмента *[–2^(2^M–1), 2^(2^M–1))* и знаку «?».

Тождественно истинные формулы такого языка будем называть формулами элементарной теории рассматриваемой сигнатуры.

Основные результаты

Теорема 1. ***P-SPACE****-полна задача проверки принадлежности формул первого порядка в рассматриваемой сигнатуре к элементарной теории этой же сигнатуры.*

Схема доказательства. Сформулированная в теореме задача может быть решена на машине Тьюринга с полиномиально ограниченной лентой, поскольку все используемые (в том числе и вспомогательные) целые числа не превосходят по абсолютной величине числа *2^(2^M–1)*. А используемая память не превосходит *m Kbyte*.

Quantified Boolean Formula (QBF), описанная в [1, 3, 8] полиномиально m-сводится к задаче из формулировки теоремы. □

Доказанная теорема допускает несколько обобщений. Первое обобщение основано на введении нескольких сортов независимых сигнатур, объединённых в одну общую сигнатуру. При этом каждая функция имеет все аргументы и результат из одного и того же носителя одной и той же независимой сигнатуры.

Второе более широкое обобщение теоремы связано с явным введением в общую сигнатуру типов аргументов и результата для каждой функции из общей сигнатуры. Каждый тип для аргументов и результата использует свой конечный носитель, элементы которого имеют имена, записанные в виде строк символов в принятой кодировке. Новые типы могут быть, например, знаком «?», или числами типа *real*, или типа *real* двойной точности, или строками из не более чем из 256 различных символов.

Формулировки и доказательства этих двух обобщений аналогичны. Они верны для программ-функций всех версий языка Паскаль.

Рассмотрим несколько другую задачу, а именно, задачу проверки Halting-problem для любой из рассматриваемых программ-функций. Назовём её H.

Проблема H:

Дано: текст программы-функции, вычисляемой на компьютере, использующем память *m Kbyte*; список значений всех её аргументов; целое положительное число *m*, записанное в унарной системе счисления.

Вопрос: Проверить, является ли программа-функция успешно завершающей свою работу над этим списком с использованием памяти *m Kbyte*.

Теорема 2. *Проблема H является **P-SPACE**-полной.*

Схема доказательства. Проверка нехватки памяти или бесконечной зацикленности выполнения программы от входных данных осуществляется посредством подсчёта числа выполняемых шагов на вводимой дополнительной памяти. Если число этих шагов превышает число внутренних различных вариантов содержимого памяти, умноженного на длину записи текста программы, то программа на этих данных бесконечно зациклена. Такую проверку можно осуществить на ленте, число ячеек которой полиномиально ограничено от длины программы.

Следовательно, задача H принадлежит **P-SPACE**. Задача QBF полиномиально m-сводится к H. □

Замечание. Проблема H остаётся P-SPACE-полной, если ограничиться переменными только типа *Boolean*.

Широко известная алгоритмически неразрешимая проблема останова для типов данных бесконечного объёма становится разрешимой и, более того, P-SPACE-полной. В результате видно, что программы-функции, вычисляемые на компьютере с фиксированной памятью, существенно меняют свои вычислительные возможности.

Для машин Тьюринга теорема, аналогичная теореме 2, доказана в [8].

Отметим, что широко используемые для обучения экспоненциальная функция, факториал и функция, вычисляющая *n*-ное число Фибоначчи не принадлежат **FP-SPACE**.

Литература

1. *Ахо А., Хопкрофт Дж., Ульман Дж.* Построение и анализ вычислительных алгоритмов. — М.: Мир, 1979.
2. *Бабаев И.О., Герасимов М.А., Косовский Н.К., Соловьев И.П.* Интеллектуальное программирование. Турбо-Пролог и Рефал-5 на персональных компьютерах. Изд. СПбГУ, 1992. 167 с.
3. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.
4. *Косовский Н.К., Косовская Т.М.* Полиномиальность и NP-трудность задачи вычисления знака постоянного терма // Математические вопросы кибернетики, вып.16, 2007. М.: Физматлит. С. 125–128.
5. *Косовский Н.К.* Полиномиально ограниченный сверху объём изменений программ на RAM+BOOL для доказательства принадлежности FP. // Исследования по конструктивной математике и математической логике XII. Сборник работ. — СПб.: С.Петербургское отделение Математического института им. В.А. Стеклова РАН, ВВМ, 2012. — С. 105–110.
6. *Форсайт Р.* Паскаль для всех. М.: Машиностроение. 1986.
7. *Яхонтов С.В., Косовский Н.К., Косовская Т.М.* Эффективные по времени и памяти алгоритмические приближения чисел и функций. Учебное пособие. СПб: Изд-во С.-Петерб. ун-та, 2012. 256 с.

8. *Du D.Z., Ko K.I.* Theory of Computational Complexity. A Wiley-Interscience Publication. John Wiley & Sons, Inc. 2000.
9. *Kosovskiy N., Kosovskaya T.* Polynomial-Time Effectiveness of Pascal, Turbo Prolog, Visual Prolog and Refal-5 Programs // International Journal «Information Models and Analyses», Vol. 1. 2012. P. 94–99.

РАНДОМИЗАЦИЯ, УСРЕДНЕНИЕ И МУЛЬТИАГЕНТНЫЕ ТЕХНОЛОГИИ В DATA MINING И УПРАВЛЕНИИ

О.Н. Граничин

Санкт-Петербургский государственный университет
198504, Санит-Петербург, Университетский пр. 28, Россия
Oleg_granichin@mail.ru
тел: +7 (812) 428-49-10, факс: +7 (812) 428-49-10

Abstract. This article is presented the key ideas and applications of the three important tools for system analysis and design, namely, randomization, averaging and multiagent technologies. These tools can largely facilitate the numerical solutions of various estimation, optimization and adaptive control problems, which are typically formulated under limited resources and insufficient data.

Ключевые слова: рандомизированные алгоритмы, мультиагентные технологии, управление, извлечение знаний.

Введение

Последние 200 лет были «колыбелью» многих больших и малых революций (рисунок 1). Промышленная революция XIX — начала XX вв. и Первая мировая война породили волну социальных революций, которые если не перевернули, то основательно потрясли мировые устои. Вторая мировая война и послевоенное развитие экономики, исследования в ядерной и микромолекулярной физике, электронике твердого тела и пограничных явлений, создание первых промышленных вычислительных устройств дали толчок индустриальной революции, которая за четверть века подготовила почву для бурного всплеска развития информационных технологий.

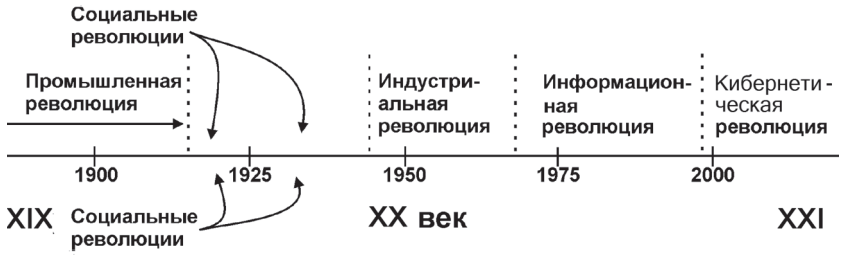


Рис. 1. Последовательность революций XIX–XXI веков

Изобретение компьютера — универсального, многофункционального, электронного авто-матического устройства для работы с данными и инфор-

мацией — привело к тому, что компьютерная техника в современном обществе взяла на себя значительную часть работ, связанных с её обработкой, систематизацией и хранением.

В XXI веке мы наблюдаем новый сдвиг в технологиях, который проявляется в разных аспектах. Если в последней четверти XX века подчеркивали, что развитие нашего общества находится на стадии урбанизации, индустриальной экономики, технологии массовых продуктов, то сейчас её все больше характеризуют глобализация, экономика знаний, цифровые технологии. Ранее ключевым ресурсом был капитал, сейчас — знания/информация. В распределении материальных объектов превалировали дороги и транспорт, сейчас на передовые позиции выходят цифровые сети. Ранее фокус был на региональной сфере, сейчас — на глобальной. Факторами успеха были возможности экономии ресурсов и сокращения затрат при масштабировании производства, теперь таким фактором стала адаптивность — способность быстрого ответа в виде эффективного решения на непредсказуемые изменения [1]. При этом за последние полвека стоимость применения цифровых технологий снизилась чрезвычайно сильно в отличие от роста затрат на труд и на использование традиционных механических технологий.

Сейчас человечество делает серьезный шаг к новой реальности: кибернетическому будущему. По многим признакам мы вступили в новую фазу кибернетической революции. На диаграмме 1 начало этой фазы показано в конце 90х годов XX века. Понять суть происходящих изменений нельзя без переосмысления многих аспектов парадигмы «Что такое вычислительное устройство и что такое вычислительный процесс». В ближайшее время должны измениться традиционные представления о том, как устроен компьютер, что такое вычислительная система. Эти процессы принесут изменения в стиль программирования, в то, как будут использоваться вычислительные устройства. Переход к новой парадигме вычислений приведет, наверное, к тому, что архитектура вычислительных устройств «сдвинется» в сторону «набора одновременно работающих асинхронных моделей взаимодействующих динамических систем (функциональных элементов)» [2]. Среди новых характерных черт будущей парадигмы все более отчетливо проступают следующие: стохастичность, гибридность, асинхронность, кластерность (отсутствие жесткой централизации и динамическая кластеризация на классы связанных моделей), мультиагентность [3].

Изменения происходят не только в парадигмах управления техническими устройствами. И в менеджменте на смену идеологии, ориентированной на «культ начальства»: руководитель — «это мыслитель, стратег, учитель» и т. д., а его окружение — лица, которые нуждаются в опеке руководителя, но сами инициативы проявлять не могут, должна прийти более гуманистическая теория организации, учитывающая реальные интересы ее участников. Один из таких новых подходов разрабатывается В.А. Виттихом [4]. Это

«теория интерсубъективного управления», которая противопоставляется «бюрократической теории». Теория исходит из того, что резервы повышения эффективности управления следует искать не в модернизации бюрократической машины, а в самих людях, в каждом человеке, в использовании его интеллектуальных и волевых ресурсов на фоне применения новейших технологий поиска, обработки и использования информации. Теория интерсубъективного управления по отношению к класической теории менеджмента имеет совершенно иные — в некотором смысле, противоположные предпосылки. Вместо менеджеров — профессиональных начальников всех рангов — появляются «солидарные акторы», осознающие проблемную ситуацию и на основе объективных данных и партнёрских отношений стремящиеся урегулировать её, соблюдая достигнутые договорённости и не прибегая ни к каким насильственным методам (в том числе, к административному давлению или принуждению). В качестве таких акторов могут выступать граждане, представители органов власти, бизнеса и общественных организаций, у которых, как правило, будут разнообразные и противоречивые интересы, но которые в определенный период времени могут быть связаны общими интересами в достижении общей цели.

В этой статье на основе анализа современного понимания понятий: *информация*, *сигналы*, *данные*, *знания* и *управление* — выдвигается положение о тесном соединении процессов обработки данных и управления, о ликвидации разграничительных барьеров между ними и приводятся примеры более эффективного решения нескольких «трудных» задач с существенными неопределенностями новыми методами.

1. Информация, сигналы, данные, знания и управление

В XX веке слово «информация» стало термином во множестве научных областей, получив особые для них определения и толкования. Обычно мы рассматриваем информацию как сообщение о том, что что-то произошло, т.е. одна из важнейших ее черт — изменение чего-то (объекта информации). Но информация возникает у кого-то или в чем-то (субъект информации).

На самом деле два крайних термина в заголовке этого раздела неразрывно друг с другом связаны и не могут друг без друга существовать. Ценность информации определяется ее способностью «подтолкнуть» субъекта к определенным действиям, т.е. его способности на основании полученной информации сформулировать некоторое управляющее воздействие. Информация, «не подталкивающая» к действию, – бессмысленна, так же как и бессмысленны какие-то действия без лежащей в их основе информации.

Н. Винер более шестидесяти лет тому назад провозгласил начало эры новой науки «Кибернетики», одним из первых четко подметив, что «информационно-управленческая связь – это существенная часть любых явлений в живой и не живой природе» [5].

Введем для информации обозначение x, а для управления — u. Процесс принятия управленческого решения формализовано может быть записан так

$$u = U(x), \tag{1}$$

где $U(\cdot)$ — некоторая функция от x. Часто информацию x отождествляют с вектором состояний исследуемой системы. В этом случае говорят, что соотношение (1) задает обратную связь по состоянию. Реализация управления естественно влияет на информацию, способствуя новым изменениям объекта информации. Сформированное управление u поступает в систему и воздействует на состояние x, во многих случаях изменяя его.

Формализация процесса принятия решений приводит к необходимости определения понятий: сигналы, данные и знания.

Информация проявляет себя через изменения в тех или иных физических или социальных явлениях. Например, у заболевшего человека повышается температура тела. При падении дерева органы слуха фиксируют изменения в окружающих нас звуках, а глаза — изменения в отображаемой картинке. При движении подводной лодки от ее гребных винтов распространяются специфические акустические волны. При приближении самолета и попадании его в зону видимости нашего радара от него отражается посылаемая радаром электромагнитная волна. Социальные изменения проявляются через выборочные опросы, голосования и т. п. Физические и социальные явления и процессы, изменение которых можно зарегистрировать с помощью органов чувств или приборов, называются *сигналами*. Результаты регистрации называются *данными*. Осознание связей между зарегистрированными данными и информацией, а также способы выбора управляющих воздействий в зависимости от той или иной информации будем называть *знаниями*.

Например, регистрация температуры тела человека дала значение более 38,5 градусов Цельсия. Это говорит о том, что пациент болен.

Обозначим через f — сигналы и через y — данные. Схема процесса регистрации сигнала показана на рисунке 2.

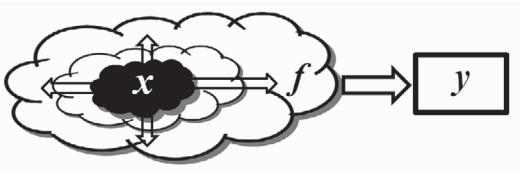


Рис. 2. Схема регистрации сигнала (получения данных)

Если нам доступны только данные наблюдений, то процесс принятия решения — это выработка управляющего воздействия на основе зарегистрированных данных

$$u = U(y). \tag{2}$$

Законы управления такого типа называются обратными связями по наблюдениям.

На практике реальная ситуация осложняется еще и тем, что при получении данных в любом регистрационном устройстве к сигналам добавляются некоторые помехи (ошибки) v. Хорошо поставленный эксперимент при тщательном измерении позволяет в некоторых случаях свести ошибки v к минимуму $v \approx 0$. Если первоначально «чистый» эксперимент не поставить, то стараются сделать его таковым с течением времени, т. е. $v \to 0$. В других случаях в задачах о наблюдении физических явлений часто достаточно обоснованно предполагают статистическую природу помех, но возможны осмысленные постановки задачи и при произвольных внешних помехах [6].

В работоспособных системах имеющиеся знания позволяют на основании получаемых данных y формировать управляющие воздействия u, которые или дают какой-то выигрыш или позволяют как-то скомпенсировать негативную информацию. Без базы знаний постановка задачи о выработке обоснованных управляющих воздействий оказывается почти бессмысленной. Уточнение «почти» включено, т. к. при априорном отсутствии знаний их зачастую можно приобрести с течением времени.

Совокупность накопленных знаний называют *онтологией*.

Знания в онтологию поступают как «из вне» в виде постулируемых законов («откровений»), так и формируются «внутри» системы при обработке данных. Процесс извлечения знаний из данных определяет круг задач такой новой бурно развивающейся области как Data Mining. В контексте задач управления онтология служит своеобразной базой данных (знаний), из которой выбирается наиболее адекватная текущей ситуации функция управления $U(\cdot)$ (см. рисунок 3).

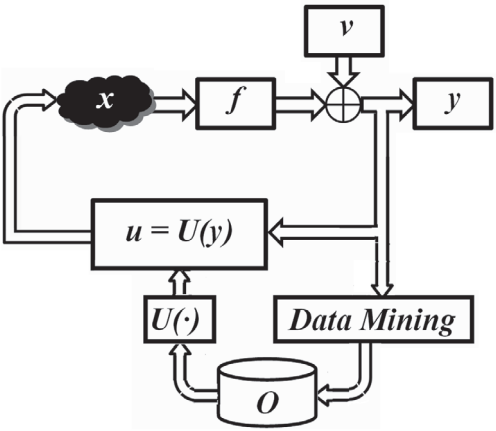


Рис. 3. Обратная связь по наблюдениям с помехами и выбор функции управления $U(\cdot)$

Изменение набора знаний со временем очень важная черта, позволяющая адаптироваться к изменяющимся условиям.

2. Соединение процессов обработки данных и управления

Процессы управления и накопления знаний часто являются взаимно противоречивыми. Целью управления обычно является достижение какого-то устойчивого состояния (по возможности не изменяющегося со временем). В этом состоянии «очень мало информации», и, следовательно, невозможно выявить или установить новые связи, значения и т. п. Например, о лежащем в пыли на обочине дороги камне мало что можно узнать при поверхностном осмотре. Его неизменность дает мало информации (характеристик изменений). Камень надо перевернуть, поднять, толкнуть или расколоть для получения какой-то информации. Это приводит к тому, что при синтезе законов управления часто сталкиваются с проблемой недостаточной вариативности последовательности наблюдений. А.А. Фельдбаум сформулировал известный принцип «дуального управления»: управляющие воздействия должны быть в известной мере изучающими, но, в известной мере, направляющими [7]. Например, если цель адаптивного управления состоит в минимизации отклонения вектора состояния системы от заданной траектории, то это часто приводит к вырожденной последовательности наблюдений, в то время как для успешного проведения идентификации неизвестных параметров системы должно быть обеспечено «разнообразие» наблюдений.

Сложившаяся к настоящему времени парадигма использования вычислительных устройств базируется на исторически сложившемся разделении процессов обработки данных и принятия управленческих решений (после обработки). Основания этого разделения прослеживаются в истории развития средств вычислительной техники. Первоначально компьютеров было мало и они, занимая огромные пространства, требовали специальных условий для эксплуатации. Формировались особые вычислительные центры для объединенного решения в одном месте множества разных задач, причем до сих пор актуальным остается одно из приоритетных направлений развития — создание суперкомпьютеров. Встроенным устройствам традиционно отводилась роль или устройств для сбора данных, или устройств для реализации определенных управляющих воздействий. В некоторых случаях они использовались как регуляторы в простых контурах обратной связи. Суперкомпьютеры брали на себя выполнение задач Data Mining.

Но надо четко отдавать себе отчет в применимости этой традиционной парадигмы. В природе и обществе все-таки информационно-управленческие связи являются основой всех явлений и процессов. Искусственно разделяя процессы обработки данных и управления, мы существенно снижаем наши потенциальные возможности использования информационно-коммуникационных технологий.

Надо ли разделять процессы обработки данных и управления?

С начала XXI века в теории управления заметен всплеск интереса к тематике управления в сетях, коллективному взаимодействию, мультиагентным технологиям и т. п. Это во многом связано с технологическим прогрессом. Сейчас миниатюризация и быстродействие средств вычислительной техники достигли такого уровня, что стало возможным в миниатюрных встроенных системах реального времени использовать вычислительные блоки, соизмеримые по производительности с мощными компьютерами XX в. Все чаще «простые» встроенные устройства заменяются на «интеллектуальные встроенные системы». Новые альтернативы позволяют по-новому взглянуть на Data Mining. В литературе все чаще появляются мысли о возрождении науки «Кибернетика» с большой буквы, о появлении «неокибернетики» [8]. Теория управления, начавшись с регуляторов механических систем в XIX в., пройдя к концу XX в. этап глубокой интеграции с цифровыми технологиями обработки данных и принятия решений, фокусируясь в XXI в. на сетях объектов, выступает «собирателем» трех основных компонент прогресса второй половины XX века: теории управления, теории коммуникаций, информатики [9].

Может ли дать какое-то новое качество в обработке данных и извлечении знаний применение кибернетической парадигмы, при которой процессы «добычи знаний» и получения информации будут учитывать неразрывную связь информации и управления (и опираться на нее)?

Да, может!!!

2.1. Эффективность замкнутых стратегий в условиях неопределенностей

Для иллюстрации положительного ответа в [10] рассмотрены несколько примеров повышения эффективности процессов обработки данных и управления при изменении парадигмы. Один из них основан на использовании замкнутых стратегий управления в условиях неопределенностей. Для объекта управления (ОУ) с входами $u(t)$ и выходами $y(t)$, и предположим, что задано начальное состояние $y(0)=1$, и динамика объекта при $t=1,2$ описывается уравнением

$$y(t)+ay(t-1)=u(t-1)+v(t)$$

с неопределенностями $v(t)$ и a двух типов:

- динамические возмущения $v(t)$ неизвестны и ограничены для всех t: $|v(t)|\leqslant 1$, но могут меняться со временем;
- коэффициент модели a также неизвестен и ограничен: $a\in[1;5]$, но он не может изменяться со временем.

Можно выбрать входы $u(0)$ и $u(1)$. Цель — минимизировать $|y(2)|$.

При сравнении качества минимаксной оптимизации

$$J = \sup_{a \in [1;5]} \sup_{|v(1)| \leqslant 1, |v(2)| \leqslant 1} |y(2)| \to \min_{u(0), u(1)}$$

для двух классов допустимых стратегий управления: программных (всевозможные пары $u(0)$, $u(1)$) и замкнутых (в которых в момент времени $t=1$ можно использовать наблюдение $y(1)$ и управление $u(0)$), получились два существенно отличающихся ответа $J_{pr}^{opt}=8{,}25$ и $J_{cl}^{opt}=2{,}125$.

Зависимость качества управления от задания класса неупреждающих стратегий адекватно понимается далеко не всех публикациях. Если все параметры объекта управления известны и помехи отсутствуют, то множества программных и замкнутых стратегий управления оказываются совпадающими.

2.2. Усреднение наблюдений и рандомизация

Другой пример из [10] показывает возможность получения обоснованных оценок неизвестных параметров системы в условиях произвольных внешних возмущениях в измерениях при допущении возможности активно влиять на результаты измерений, выбирая рандомизированные управления (входы).

В случае больших и сложных систем, состоящих из похожих компонентов, в статистической механике и физике оправдал себя подход Крылова-Боголюбова, основанный на усреднении данных, активно развиваемый, начиная с работ Гиббса, основываясь на теории Лебега. Для большого количества физических и социальных явлений при отсутствии внешних воздействий выполняется гипотеза эргодичности, когда среднее пространственное значение той или иной характеристики различных компонент системы, подсчитанное в определенный момент времени, совпадает со средним временным значением одной из компонент. При этом идеи усреднения хорошо согласуются и с конструкцией многих регистрирующих приборов, принцип действия которых часто заключается в том, что они выдают в результате некоторое среднее значение той или иной характеристики за определенный интервал времени.

Если регистрирующий прибор усредняет поступающие при $t=1, \ldots, T$ «мгновенные» сигналы $f(t)$ с помехами $v(t)$, то на выходе прибора получаем

$$y = \sum_{t=1}^{T} f(t) + \sum_{t=1}^{T} v(t).$$

При случайной природе помех $v(t)$ и известном среднем значении m в предположении об их независимости, одинаковой распределенности, конеч-

ности дисперсии, в силу закона больших чисел теории вероятностей, увеличивая T, можно добиться сколь угодно малой вероятности отличия второго слагаемого в последней формуле от среднего значения помехи. Т. е. по данным y и m можно достаточно точно определить среднее значение сигнала $f(t)$, определяемое первым слагаемым в последней формуле.

Что делать, если помехи $v(t)$ не являются случайными (статистическими)? Например, $v(t)$ — значения неизвестной функции (т. е. произвольные).

В рамках классической парадигмы обработки данных постановка задачи об оценивании среднего значения сигнала регистрируемого на фоне произвольных помех кажется абсурдной, но не из-за ее практической бессмысленности (это очень важная задача), а из-за невозможности ее как-то решить.

Для простоты рассмотрим случай скалярных наблюдений. Модернизируем постановку задачи, включив в модель наблюдений управляющее воздействие (вход) u.

Следуя парадигме неразрывности информации и управлений, будем считать, что регистрируемый сигнал f в момент времени t напрямую определяется текущим входом $u(t)$ и некоторым неизвестным параметром x (неизвестным коэффициентом усиления/ослабления входа)

$$f(t) = x \cdot u(t).$$

Модель наблюдений можно переписать в виде:

$$y(t) = x \cdot u(t) + v(t), \quad t = 1, 2, \ldots, T. \tag{3}$$

Причем мы можем

- выбирать входы $u(t)$,
- измерять выходы $y(t)$.

При использовании $u(t) \equiv 1$ получаем традиционную задачу об оценивании неизвестного параметра x, наблюдаемого на фоне помех.

На самом деле с такими постановками задач мы сталкивались уже в школьной программе изучая физические эксперименты, в которых измеряется результат того или иного воздействия на систему. Например, прикладывая к пружине разные усилия, мы получаем разные длины растяжения или сжатия. Но полученные результаты — это не произвольные числа, они определяются характеристиками (параметрами) самой пружины (коэффициентом упругости). Кроме того, на результат влияют и конкретные условия проведения эксперимента, определяемые внешними силами, помехами и т. п. (в частности, трение).

Другой типичный пример из области исследования материалов, или недр, или разнообразных применений дистанционного зондирования показан на рисунке 4. Источник посылает в пространство некоторый сигнал:

или поток частиц (электроны, alpha-частицы и т. п.), или волну той или иной природы (акустической или электро-магнитной), — причем интенсивность потока можно варьировать и точно измерять (например, мерить величину «ушедшего» заряда). При встрече с мишенью либо сигнал отражается (эхо-акустика, радар и т. п.), либо его взаимодействие с мишенью порождает некоторый новый «отраженный» (может быть и другой природы), интенсивность которого потом измеряется детектором. Требуется определить неизвестный параметр x, который либо равен нулю, что соответствует отсутствию мишени с заданными отражающими свойствами, либо при положительной величине x характеризует те или иные физические характеристики: дальность до мишени, ее размер, отражающие свойства материала и т. п.

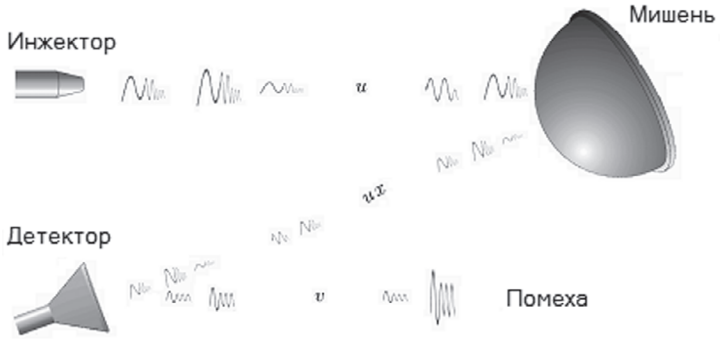


Рис. 4. Источник, мишень и детектор отраженного сигнала

Общим в рассмотренных примерах является возможность активного влияния экспериментатора на результат наблюдений (возможность подать «входное воздействие»).

В каком максимально широком классе помех $v(t)$ все-таки реалистично попытаться получить осмысленный ответ в задаче об оценивании неизвестного параметра x?

Естественно при записи уравнения (3) предположить, что второе слагаемое в правой части $v(t)$ включает в себя все неопределенности влияющие на выход $y(t)$, которые никак не связаны с $u(t)$, т. к. $u(t)$ явно входит только в первое слагаемое в правой части (3). Такие помехи будем называть «внешними», подчеркивая их независимость от внутренних входов, подаваемых в систему. Можно ли как-то решить задачу в таком классе неопределенностей?

Поясним схему задачи, используя рисунок 5. Система является «черным ящиком» с входом $u(t)$ и выходом $y(t)$. Система характеризуется неизвестным нам параметром x (в примере с пружиной x — это коэффициент упругости). Экспериментатор может выбирать воздействия на систему $u(t)$, которые поступают на вход «черного ящика» (в примере с пружиной мы мо-

жем растянуть или сжать ее на расстояние $u(t)$). На выходе «черного ящика» к результату добавляется внешняя помеха $v(t)$, которая никак не связана с «внутренними» процессами внутри «черного ящика» (в примере с пружиной выход с помехой — погрешности измерений, вносимые динамометром).

Рис. 5. Модель наблюдения сигнала с помехой

Уточним задачу. *Требуется* по последовательности входов и выходов $\{u(t), y(t)\}$ определить неизвестный параметр x при отсутствии каких-либо ограничений на последовательность внешних помех $\{v(t)\}$.

Не кажется ли такая постановка задачи абсурдной?

С детерминистской точки зрения — конечно! Не может быть никакого детерминированного алгоритма, дающего хотя бы в каком-то смысле здравый ответ (кроме бессмысленного решения — вся числовая ось). Предложив в качестве ответа любое из чисел или даже какой-то интервал при конечном (или счетном) числе наблюдений, всегда можно будет подобрать такие $v(t)$, что при следующем наблюдении предложенный ответ будет неверным.

Общий алгоритм последовательного оценивания неизвестного параметра x состоит из двух шагов:

1. Выбор входа $u(t)$.
2. Оценивание параметра x на основе полученных данных $u(t)$, $y(t)$ (например, вычисление числовой оценки $\hat{x}$ или множества $\hat{X}$, содержащего x).

Если бы в условиях задачи дополнительно можно было бы предположить случайную (вероятностную) природу помех $v(t)$, то при выполнении условий закона больших чисел можно было бы говорить об оценивании неизвестного параметра x путем простого усреднения данных наблюдения.

Если наблюдения проводить также со случайной помехой, но у которой среднее значение m было бы неизвестно, то результаты усреднения отличались бы от истинного значения x, на неизвестную величину m.

Несмотря на кажущуюся абсурдность постановки задачи оценивания при произвольных внешних помехах, из практических потребностей часто ее все-таки приходится решать.

Альтернативой оказываются *рандомизированные алгоритмы, в которых выполнение одного или нескольких шагов производимых пользователем основано на случайном правиле (т. е. среди многих детерминированных правил одно выбирается случайно в соответствии с вероятностью P).*

В зависимости от специфики конкретной задачи вероятность P или является искусственным элементом, вводимым в алгоритм для улучшения разрешимости проблемы, или в рассматриваемой системе могут присутствовать измеряемые случайные элементы. Выбор этой вероятности P является частью конструирования алгоритма.

Рассмотрим следующее правило случайного выбора для первого шага рандомизированного алгоритма последовательного оценивания неизвестного параметра x

$$u(t)=\begin{cases} +1 \text{ с вероятностью } ½, \\ -1 \text{ с вероятностью } ½. \end{cases}$$

На втором шаге по известным парам значений $(u(t), y(t))$ формируем величины $\overline{y}(t)=u(t)\cdot y(t)$. Для «новой» последовательности наблюдений справедлива похожая на исходную модель

$$\overline{y}(t)=x\cdot\overline{u}(t)+\overline{v}(t),$$

в которой $\overline{u}(t)=u(t)^2=1$, $\overline{v}(t)=u(t)\cdot v(t)$.

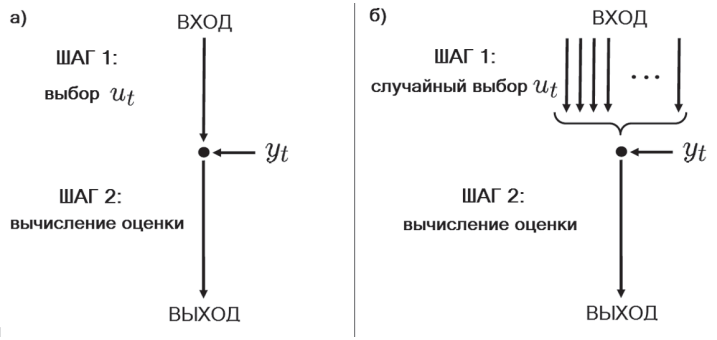


Рис. 6. Детерминированный (а) и рандомизированный (б) алгоритмы

Если $v(t)$ — внешние помехи, то естественно считать, что они независимы с нашим рандомизированным правилом выбора входов на шаге 1. Следовательно,

$$E\overline{v}(t)=Eu(t)\cdot v(t)=Eu(t)\cdot Ev(t)=0\cdot Ev(t)=0,$$

т. е. «в новой модели» наблюдений задача об оценивании неизвестного параметра x, не имевшая решения, превращается при использовании слу-

чайного правила выбора входов на шаге 1 рандомизированного алгоритма в «стандартную» задачу об оценивании неизвестного параметра x, наблюдаемого на фоне независимых центрированных помех.

В [10, 11] приведены алгоритмы, обоснования и примеры численного моделирования, дающие при фиксированном малом ($T=7$) количестве наблюдений доверительные интервалы для x с задаваемой априорно вероятностью p.

Итак, для, казалось бы, абсурдной задачи об оценивании параметра при произвольных внешних помехах, с которой принципиально не может справиться ни один детерминированный алгоритм, внесение рандомизации в процесс выбора входов дает получить вполне осмысленные результаты, позволяя говорить о вероятностной успешности рандомизированного алгоритма с некоторым параметром (вероятностью) p. Достижение успешных результатов с высокой степенью вероятности, в отличие от детерминированного случая, соответствует компромиссу: если полностью гарантированный результат получить невозможно, то лучше иметь какую-то гарантию, чем не иметь ничего.

Конечно, не во всех задачах компромисс возможен. Во многих случаях нужен ответ, гарантированный на 100%. Но, «защищая» рандомизированные алгоритмы, надо отметить, что уровень достоверности p обычно является параметром алгоритма, который может быть настроен пользователем. Параметр p ослабляет понятие детерминированной разрешимости, для которой вероятность успеха может быть только 0 или 1 — образно выражаясь, результат «черный» или «белый». Переходя к рандомизированным алгоритмам, p становится непрерывным параметром, пробегающим интервал $[0; 1]$, задавая тот или иной оттенок «серого».

Замечание. Альтернативный вероятностный подход к решению задачи оценивания — байесовский, при котором присутствующим в системе помехам $v(t)$ априори приписывается вероятностная природа Q, но его невозможно применить при произвольных внешних помехах (в худшем случае), т. к. все выводы имеют вероятностную основу предположений о системе. По смыслу байесовский и рандомизированний подход совершенно различны с практической точки зрения. В байесовском — Q описывает вероятность того или иного значения помехи $v(t)$ по сравнению с другими, т. е. выбор Q является частью модели задачи. В отличие от этого вероятность P в рандомизированном подходе является тем, что мы искусственно выбрали и используем. P существует только в нашем алгоритме, и, следовательно, нет традиционной проблемы плохой модели, как это может случиться с Q при байесовском подходе.

2.3. Мультиагентные технологии

На практике очень часто оказывается, что классические методы решения задач либо неприменимы к реальной жизни (не трудно представить

себе, что значит попытаться решить задачу управления предприятием в непредсказуемой динамичной обстановке современного бизнеса), либо они требуют огромных объемов расчетов (для которых не хватит мощности всех современных компьютеров), либо они вовсе отсутствуют. В многих таких случаях альтернативой оказываются мультиагентные технологии, суть которых заключается в принципиально новом методе решения задач. В отличие от классического способа, когда проводится поиск некоторого четко определенного (детерминированного) алгоритма, позволяющего найти наилучшее решение проблемы, в мультиагентных технологиях решение получается автоматически в результате взаимодействия множества самостоятельных целенаправленных программных модулей — так называемых агентов [12].

Как и в двух предыдущих примерах, одной из важнейших характеристик мультиагентных технологий является отказ от традиционной для информационных технологий парадигмы разделения процессов получения информации и принятия управленческих решений. В случае сложных систем, состоящих из огромного числа взаимодействующих динамических объектов, возможность получения реальной «мгновенной картины мира» можно вообразить себе только теоретически, на практике во время сбора всей необходимой информации «картина мира» может существенно измениться. При мультиагентных технологиях компоненты системы начинают взаимодействовать и реализовывать те или иные управляющие воздействия самостоятельно, не дожидаясь «команды из центра». Оказывается, что во многих практических задачах такая парадигма позволяет эффективно управлять системами в то время, как задача о сборе всей информации может так и оставаться не решенной. В [13,14] с помощью компьютерного моделирования это показано для задачи планирования загрузки автопарка грузовиков в реальном времени при динамических изменениях, в [15,16] кроме иллюстративных примеров моделирования для задачи балансировки загрузки узлов динамической сети даны и теоретические оценки оптимальности применения децентрализованной мультиагентной технологии управления загрузкой.

Задачи управления и распределенного взаимодействия в сетях динамических систем привлекают в последнее десятилетие внимание все большего числа исследователей. Во многом это объясняется широким применением мультиагентных систем в разных областях, включая автоматическую подстройку параметров нейронных сетей распознавания, балансировку загрузки узлов вычислительных сетей, управление формациями, роение, распределенные сенсорные сети, управление перегрузкой в сетях связи, взаимодействие групп БПЛА, относительное выравнивание групп спутников, управление движением групп мобильных роботов, синхронизации в энергосистемах и др. Для многих распределенных систем, выполняющих определенные действия параллельно, актуальна задача разделения пакета

заданий между несколькими вычислительными потоками (устройствами). Подобные задачи возникают не только в вычислительных сетях, но также и в производственных сетях, сетях обслуживания, транспортных, логистических сетях и др. Оказывается, что при естественных ограничениях на связи, децентрализованные стратегии способны эффективно решать такого типа задачи. Решение таких задач существенно усложняется при практическом применении из-за изменчивости структуры связей, обмене неполной информацией, которая, кроме того, обычно измеряется с помехами, а также из-за эффектов квантования (дискретизации), свойственных всем цифровым системам.

Для группы взаимодействующих агентов, обменивающихся с задержкой неполной информацией в дискретные моменты времени, при изменяющейся топологии связей для решения задачи о достижении консенсуса в [15, 16] предложен и обоснован алгоритм стохастической аппроксимации с убывающим размером шага, который позволяет каждому агенту получать информацию о состоянии своих соседей при одновременном снижении воздействия помех. Суть алгоритма – в децентрализованной балансировке. Каждый агент принимает решение о перераспределении заданий «с соседями» только на основании текущей оперативной информации о своей и их загруженности. Типичное время, за которое вся система приходит к «оптимальной загрузке» соизмеримо со временем, которое было бы затрачено на получение всей информации о загруженности сети. При классическом централизованном подходе далее надо было бы решить очень сложную задачу об оптимальном расписании, а потом еще и перераспределить задания, НО за это время новые задания и изменения в окружающей среде (топологии связи) могут сделать всю эту работу напрасной.

Заключение

Призыв: «Быстрее, Выше, Сильнее» в компьютерной области трансформируется в лозунг: «Быстрее, Мощнее, Миниатюрнее»! На самом деле эти противоречивые цели стыкуются в одном: технологии приближаются к созданию «мобильного» искусственного интеллекта. Автор надеется, что детальное и широкое обсуждение новых идей и подходов, изложенных в статье, позволит «проложить» им путь к практическому использованию.

Благодарности

Выражаю признательность В.А. Виттиху и П.О.Скобелеву за плодотворные дискуссии по современным проблемам управления и «добывания» знаний.

Работа выполнена при финансовой поддержке РФФИ (грант № 13-07-00250-а).

Литература

1. *Rzevski, G*. Modelling large complex systems using multi-agent technology // In Proc. of 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD2012), August 8-10, Kyoto, Japan, 2012, pp. 434–437.
2. *Нариньяни А.С*. Модель или алгоритм: новая парадигма информационных технологий. Российский НИИ ИИ. 2004.
3. *Граничин О.Н*. Характеристики перспективных принципиально новых компьютерных устройств и систем // Механика, управление и информатика. 2011. №5, с. 147–161.
4. *Виттих В.А*. Введение в теорию интерсубъективного управления. — Самара, Самарский научный центр РАН, 2013. — 64 с.
5. *Винер Н.* Кибернетика, или управление и связь в животном и машине. — М.: Наука. 1983.
6. *Граничин О.Н., Поляк Б.Т*. Рандомизированные алгоритмы оценивания и оптимизации при почти произвольных помехах. — М.: Наука. 2003.
7. *Фельдбаум А.А.* О проблемах дуального управления // В кн.: Методы оптимизации автоматических систем. — М.: Наука. 1972.
8. *Соколов Б.В., Юсупов Р.М.* Неокибернетика — возможности и перспективы развития // Докл. на общем пленарном заседании 5-й науч. конф. Управление и информационные технологии (УИТ-2008), Россия, СПб., 14–16 октября, 2008 г. СПб., 2008, с. 1–15.
9. *Андриевский Б.Р., Матвеев А.С, Фрадков А.Л*. Управление и оценивание при информационных ограничениях: к единой теории управления, вычислений и связи // Автоматика и телемеханика. 2010. №4, с. 34–99.
10. *Граничин О.Н.* Обратные связи, усреднение и рандомизация в управлении и извлечении знаний // Стохастическая оптимизация в информатике. 2012. Том 8. Вып. 2, с. 3–48.
11. *Граничин О.Н*. Неасимптотическое доверительное множество для параметров линейного объекта управления при почти произвольных помехах // Автоматика и телемеханика, 2012, №1, с. 24–35.
12. *Виттих В.А., Скобелев П.О*. Метод сопряженных взаимодействий для управления распределением ресурсов в реальном масштабе времени // Автометрия. 2009. Т. 45. №2, с. 84–86.
13. *Амелина Н., Лада А., Майоров И., Скобелев П., Царев А*. Исследование моделей организации грузовых перевозок с применением мультиагентной системы адаптивного планирования грузовиков в реальном времени // Проблемы управления. 2011. №6.
14. *Granichin O., Skobelev P., Lada A., Mayorov I., Tsarev A*. Comparing adaptive and non-adaptive models of cargo transportation in multi-agent system for real time truck scheduling // In: Proc. of the 4th International Conference on Computational Intelligence, pp. 282–285 (Evolutionary Computation Theory and Applications, ECTA'2012, October 5–7, 2012, Barcelona, Spain).

15. *Амелин К.С., Амелина Н.О., Граничин О.Н., Корявко А.В*. Применение алгоритма локального голосования для достижения консенсуса в децентрализованной сети интеллектуальных агентов // Нейро-компьютеры: Разработка, Применение, 2012, № 11, с. 39–47.
16. *Амелина Н.О., Фрадков А.Л.* Приближенный консенсус в стохастической динамической сети с неполной информацией и задержками в измерениях // Автоматика и Телемеханика, 2012, № 11, с. 6–30.

СОДЕРЖАНИЕ

Терехов А. Н. Хорошая технология делает трудную задачу простой.....3
Думшева Т. Д., Кандоба И. Н., Костоусова Е. К., Ложников А. Б. О задаче оптимального выведения ракеты-носителя на околоземную орбиту 14
Починский В. И., Котляров В.П., Дробинцев П.Д. Технология гидов для решения проблемы взрыва состояний….....39
М.А. Герасимов. Последняя книга Святослава Сергеевича Лаврова 50
В.В. Прохоров. Микроконтекстно-ориентированный подход к компонентному построению языков 54
Тулупьев А. Л. Алгебраические байесовские сети и родственные модели знаний с неопределенностью.....67
Штейнберг Б.Я. Оптимизация использования кэш-памяти в вычислительных программах и оптимизирующей компиляции 78
Сафонов В.О. Исследования и преподавание в области современных технологий и инструментов программирования и облачных вычислений.....84
Н.К. Косовский. Алгоритмическая сложность разрешимости языка свойств программ-функций на компьютере 89
О.Н. Граничин. Рандомизация, усреднение и мультиагентные технологии в Data Mining и управлении 98

Научное издание

Лавровские чтения 2013

Материалы
всероссийской научной конференции
по проблемам информатики

Пленарные заседания

23–26 апр. 2013 г.,
Санкт-Петербург

Компьютерная верстка:
Клочков М.Н.

9 785906 670021

Формат 60×90 1/16. Бумага офсетная.
Гарнитура Таймс. Усл. печ. л. 7,25.
Тираж 300 экз. Заказ № 1584.

Отпечатано в типографии ООО «ИПК «Береста».
196084, Россия, Санкт-Петербург
ул. Коли Томчака, 28
тел. 388-90-00
e-mail: beresta@mail.wplus.net