МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМ. М. В. ЛОМОНОСОВА

**Факультет
вычислительной математики
и кибернетики**

Сборник статей молодых ученых факультета ВМиК МГУ

Выпуск 5

МОСКВА — 2008

УДК 517.6+519.8
ББК 22
С23

Составители:
А. В. ИЛЬИН, В. В. ФОМИЧЕВ, А. В. СТОЛЯРОВ

С23 **Сборник статей молодых ученых факультета ВМиК МГУ** / Ред. совет: Ложкин С. А. и др. — М.: Издательский отдел факультета ВМиК МГУ (лицензия ИД №05899 от 24.09.2001), выпуск №5, 2008 — 133 стр.

В настоящий сборник вошли статьи, выполненные молодыми учеными факультета Вычислительной математики и кибернетики Московского государственного университета им. М.В. Ломоносова в 2007–2008 г.г.

УДК 517.6+519.8
ББК 22

ISBN 978-5-89407-340-8

СОДЕРЖАНИЕ

Данный выпуск посвящается столетию
академика
Льва Семёновича ПОНТРЯГИНА

УДК 681.3.06

ВВЕДЕНИЕ В ЗАДАЧУ ДЕКОМПИЛЯЦИИ

© 2008 г. К. Н. Долгова, В. Ю. Антонов

katerina@ispras.ru

кафедра системного программирования

1 Введение

В настоящее время в комплексном программном обеспечении часто применяются программные приложения, разработанные сторонними производителями. В ряде случаев такие приложения предоставляются без исходного кода на языке высокого уровня, необходимого для их аудита с точки зрения информационной безопасности их использования. Несмотря на это, такие приложения обязательно должны быть исследованы для оценки рисков их использования. В качестве одного из инструментальных средств для проведения такого рода анализа может использоваться декомпилятор.

Под *декомпилятором* мы будем понимать инструментальное средство, получающее на вход программу на языке ассемблера и выдающее на выход эквивалентную ей программу на некотором языке высокого уровня. В частности, на практике особую значимость имеют декомпиляторы, транслирующие ассемблерный листинг в язык Си. Такая задача актуальна, потому что ассемблерный листинг не позволяет с приемлемыми трудозатратами оценить взаимосвязь элементов программы, а также идентифицировать в программе стандартные алгоритмические конструкции, в то время как наличие восстановленной программы на языке высокого уровня дает возможность преодолеть указанные выше трудности. В качестве выходного языка у декомпиляторов наиболее часто используют язык Си, в силу того, что с одной стороны язык Си широко применяется в промышленном программирования, а с другой стороны — это процедурный язык. Восстанавливать программу из ассемблера в объектно-ориентированный язык принципиально сложнее, да и к тому, же программа, реализованная на процедурной парадигме программирования, может быть переведена в объектно-ориентированную программу посредством рефакторинга ее кода. Следовательно, в данной работе ограничим множество рассматриваемых декомпиляторов декомпиляторами, восстанавливающими программы, представленные либо на языке ассемблера, либо в виде исполняемых файлов, в язык Си.

Задача декомпиляции была поставлена в 60-е годы XX века сразу же, когда стали широко применяться компиляторы с языков высокого уровня, но не утратила своей актуальности и по сей день [2]. Эта задача не решена в полной мере до сих пор в силу ряда трудностей принципиального характера. В частности, при компиляции программы из языка высокого уровня в язык ассемблера характерно отображение «многие к одному» концепций языка высокого уровня в концепции языка ассемблера, и, как следствие, однозначное восстановление программы на языке высокого уровня становится зачастую невозможным.

В силу указанных выше причин полностью автоматический декомпилятор реализовать принципиально невозможно. Поэтому системы декомпиляции программ должны работать во взаимодействии с аналитиком, который (зачастую методом проб и ошибок) управляет процессом декомпиляции. В ходе декомпиляции программы решаются следующие задачи: выделение структурных единиц программы, в частности, подпрограмм в однородном ассемблерном листинге, выявление параметров подпрограмм и возвращаемых ими значений, структурный анализ, то есть восстановление операторов циклов, ветвлений и т. п., восстановление типов данных как базовых, так и производных и другие. В силу того, что все эти задачи достаточно трудоемки и алгоритмически неразрешимы, на сегодняшний момент нет известных декомпилятров, восстанавливающих программы в язык Си, которые качественно справляются со всеми перечисленными выше задачами. Для решения задач посредством использования декомпилятров требуется хорошо представлять возможности используемого инструмента и для достижения наилучшего результата, возможно, потребуется использовать набор декомпилятров в некоторой композиции. В данной работе предлагается обзор наиболее известных декомпилятров в

язык Си из бинарных файлов, рассматривается набор тестов, по которому можно сделать сравнительный анализ работоспособности декомпилятров и выполняется этот анализ.

В данной работе в качестве процессорной архитектуры, с которой ведётся декомпиляция, выбрана архитектура ia32, наиболее распространённая в настоящее время. В листингах фрагментов программ на языке ассемблера используется синтаксис AT&T [3].

Предлагаемая работа имеет следующую структуру. В силу того, что как в литературе, так и на практике зачастую смешивают понятия дизассемблирования программы и декомпиляции программы, то казалось бы уместным рассмотреть различия этих задач. Этому посвящен второй раздел статьи. В третьем разделе статьи дается описание основных подзадач декомпиляции с описанием возникающих трудностей при их решении. Четвертый раздел посвящен описанию существующих декомпилятров. В пятом разделе представлены результаты сравнительного тестирования декомпиляторов на разработанном наборе тестовых примеров. В заключении сформулированы выводы работы и направления дальнейших исследований.

2 Декомпиляция и дизассемблирование

Рассмотрим независимо друг от друга задачу дизассемблирования и задачу декомпилирования программ. Под декомпиляцией понимается построение программы на языке высокого уровня, эквивалентной исходной программе на языке низкого уровня (языке ассемблера). А под дизассемблированием понимается построение программы на языке ассемблера, эквивалентной исходной программе в машинном коде. Программа в машинном коде представляется либо в виде исполняемого модуля в стандартном для целевой операционной системы формате (например, для Win32 в формате PE [16], а для Linux – в формате ELF [15]), либо в виде дампа содержимого памяти, либо в виде трассы исполнения программы.

Однако традиционно задача декомпиляции рассматривается в более широком смысле, а именно, задача построение программы на языке высокого уровня по программе в машинном коде. Очевидно, что в такой постановке задача декомпиляции поглощает задачу дизассемблирования. Такое «широкое» понимание задачи декомпиляции излишне в силу того, что дизассемблирование и декомпиляция решают разные по сути задачи, хотя и используют схожие методы (в частности, построение графа потока управления и построение исполняемого покрытия программы). Так при дизассемблировании выполняется трансляция исполняемого файла, представляемого в виде набора машинных команд в программу на языке ассемблера. Язык ассемблера – это низкоуровневый язык и программы, написанные на языке ассемблера, сильно приближены по структуре к исполняемым файлам в машинном коде. При декомпиляции программа с представления низкого уровня транслируется в представление высокого уровня. Дальнейшим уровнем повышения абстракции программы может быть рефакторинг, посредством которого из программы на языке Си можно получить программу на языке Си++, например.

Рассмотрим разбиение задач декомпиляции и дизассемблирования на подзадачи. Так, при дизассемблировании должны быть решены следующие основные задачи:

1. Разделение кода и данных. Для каждой ячейки программы (или ячейки памяти дампа) должно быть установлено, хранит ли ячейка исполняемые инструкции или данных. Задача эта сама по себе алгоритмически неразрешима [7] и не всегда может быть решена однозначно (например, в случае самомодифицирующегося кода, динамически подгружаемого кода и т. п.).

2. Замена абсолютных адресов на символические.

При декомпиляции должны быть решены следующие основные задачи:

1. Выделение функций в потоке инструкций.

2. Выявление параметров и возвращаемых значений.

3. Восстановление структурных конструкций языка высокого уровня.

4. Замена всех обращений к памяти на конструкции языка высокого уровня (в частности, сюда входит идентификация обращения к локальным переменным и параметрам и их замена на символические имена, идентификация обращений к массивам и их замена на операции с массивами и т. д.).

5. Восстановление типов выявленных на предыдущем шаге объектов языка высокого уровня.

В дальнейшем мы будем рассматривать задачу декомпиляции в узкой постановке, то есть как задачу трансляции программы из программы на языке низкого уровня, в частности на языке ассемблера, в программу на языке высокого уровня, в частности на Си.

3 Обзор основных подзадачи декомпиляции

Рассмотрим основные задачи декомпиляции и подходы к их решению.

3.1 Выделение функций

Одной из основных структурных единиц программ на языке Си являются функции, которые могут принимать параметры и возвращать значения. Откомпилированная программа, однако, состоит из потока инструкций, функции в котором никак структурно не выделяются. Как правило, компиляторы генерируют код с одной точкой входа в функцию и одной точкой выхода из функции. При этом в начало кода для функции помещается последовательность машинных инструкций, называемая прологом функции, а в конец кода, генерируемого для функции, помещается эпилог функции. И пролог, и эпилог функции, как правило, стандартны для каждой архитектуры, и претерпевают на ней незначительные вариации. Например, стандартный пролог и эпилог функции для архитектуры i386 показаны на рис. ниже:

Пролог:

```
pushl   %ebp
movl    %esp, %ebp
```

Эпилог:

```
movl    %ebp, %esp
popl    %ebp
ret
```

Прологи и эпилоги функций легко могут быть выделены в потоке инструкций. Кроме того, при работе с трассами можно считать, что инструкции, на которые управление передается с помощью инструкции `call`, являются точками входа в функции, а инструкции ret завершают функции.

Возникает соблазн считать инструкции, расположенные между прологом и эпилогом, или между точкой входа и выходом телом функции, однако в этом случае можно натолкнуться на ряд сложностей.

Во-первых, при компиляции программы могут быть указаны опции, влияющие на форму пролога и эпилога функции. Например, опция компилятора GCC `-fomit-frame-pointer` подавляет использование регистра `%ebp` в качестве указателя на текущий стековый кадр в случаях, когда это возможно. В этом случае пролог и эпилог функции будут как таковые отсутствовать.

Во-вторых, отдельные оптимизационные преобразования могут разрушать исходную структуру функций программы. Очевидным примером такого оптимизационного преобразования является встраивание тела функции в точку вызова. Встроенная функция не существует как отдельная структурная единица программы, и ее автоматическое выделение представляется затруднительным.

Существуют оптимизирующие преобразования, которые приводят к появлению в машинном коде конструкций, принципиально невозможных в языках высокого уровня. Таким оптимизирующим преобразованием является, например, sibling call optimization. Если список параметров двух функций идентичен, и первая функция вызывает вторую с этими параметрами, то инструкция вызова подпрограммы `call` может быть преобразована в инструкцию безусловного перехода jmp в середину тела второй функции. Результатом такого рода «неструктурных» оптимизаций будет появления переходов из одной функции в другую, появление функций с несколькими точками входа или несколькими точками выхода. Другим источником таких «неструктурных» конструкций в машинной программе являются операторы обработки исключений таких языков, как Си++.

Таким образом, хотя в «обычном» случае компилятор генерирует хорошо структурированный код, поддающийся разбиению на функции, достаточно легко может быть получен и «неструктурированный» код. Следует отметить, что в этом случае влияние программиста, пишущего на языке Си, на структуру генерируемого кода достаточно ограничено возможностями языка Си, не позволяющего бесконтрольной передачи управления между функциями и не поддерживающего механизм исключений. Однако можно предполагать, что если восстанавливается программа с языка ассемблера, полученная либо в результате компиляции программы на языке Си или в результате работы дизассемблера, то она не содержит «неструктурных» особенностей, описанных выше, и, таким образом, может быть разбита на функции.

3.2 Выявление параметров и возвращаемых значений

Языки высокого уровня, в частности, Си поддерживают передачу параметров в функции и возврат значений. В языке Си существует только передача параметров по значению, в других языках могут поддерживаться и другие механизмы. Заметим, что здесь мы рассматриваем только механизмы передачи параметров, отображаемые в генерируемый машинный код. Передача параметров по имени, передача параметров в шаблоны и другие механизмы периода компиляции программы здесь не рассматриваются.

Способы передачи параметров и возврата значений для каждой платформы специфицированы и являются составной частью так называемого ABI (application binary interface). Под платформой здесь понимается, как обычно, тип процессора и тип операционной системы, например, Win32/i386 или Linux/x86_64. Одной из задач ABI является обеспечение совместимости по вызовам приложений и библиотек, скомпилированных разными компиляторами одного языка или написанных на разных языках.

Так, для платформы win32/i386 используется несколько соглашений о передаче параметров. Соглашение о передаче параметров `_cdecl` используется по умолчанию в программах на Си и Си++ и имеет следующие особенности [9]:

1. Параметры передаются в стеке и заносятся в стек справа налево (то есть, первый в списке параметр заносится в стек последним).

2. Параметры выравниваются в стеке по границе 4 байт, и адреса всех параметров кратны 4. То есть параметры типа `char` и `short` передаются как `int`, но и дополнительное выравнивание для размещения, например, `double` не производится.

3. Очистку стека производит вызывающая функция.

4. Регистры `%eax`, `%ecx`, `%edx` и `%st(0)`–`%st(7)` могут свободно использоваться (не должны сохраняться при входе в функцию и восстанавливаться при выходе из нее).

5. Регистры `%ebx`, `%esi`, `%edi`, `%ebp` не должны модифицироваться в процессе работы функции.

6. Значения целых типов, размер которых не превосходит 32 бит возвращаются в регистре `%eax`, 64-битных целых типов — в регистрах `%eax` и `%edx`, вещественных типов — в регистре `%st(0)`.

7. Если функция возвращает результат структурного типа, место под возвращаемое значение должно быть зарезервировано вызывающей функций. Адрес этой области памяти передается как (скрытый) первый параметр.

Отметим, что этот набор правил — это именно соглашения, которые «добровольно» выполняются в сгенерированном коде. Пока речь не заходит об интерфейсе с независимо скомпилированными сторонними модулями программист может в определенной мере модифицировать эти правила, существенно затрудняя задачу автоматического восстановления функций.

Опять же можно предполагать, что если программа транслируется из автоматически полученного ассемблерного кода (либо компилятором, либо дизассемблером), то в ней используются только соглашения о передаче параметров из некоторого предопределенного множества. Причем в одной программе для разных функций не могут использоваться разные соглашения о передаче параметров.

На первом этапе решения задачи выявления параметров функций следует определить следующие особенности вызова функций:

1. Используемое соглашение о передаче параметров (выбрать одно соглашение из набора предопределенных соглашений).

2. Размер области параметров функции. Большинство соглашений о передаче параметров могут быть достаточно надежно идентифицированы по используемым инструкциям. Так, соглашение о передаче параметров `_stdcall` требует, чтобы параметры из стека удалялись вызываемой функцией. Для этого может использоваться единственная инструкция системы команд i386 — `ret N`, где N — размер удаляемых из стека параметров. Таким образом, использование этой инструкции для возврата из функции указывает как на соглашение о передаче параметров, так и размер параметров функции.

В случае вызова функции по указателю при статическом анализе нам может быть неизвестен адрес вызываемой функции. В этом случае отследить, как возвращается управление из вызываемой функции, не представляется возможным. Определение соглашения о вызовах тогда должно быть отложено на фазы последующего анализа.

Итак, на фазе выявления параметров и возвращаемых значений определяется размер передаваемых в функцию параметров и способ возврата значения из функции. В дальнейшем эта информация используется как начальная при восстановлении символических имен и восстановлении типов.

3.3 Структурный анализ

Одним из результатов предыдущих фаз анализа ассемблерного листинга программы является разбиение потока инструкций ассемблерного листинга на отдельные функции и выявление точек входа в функции и возврата из функций.

Инструкции ассемблерной программы в функции могут рассматриваться как представление нижнего уровня (Low-level intermediate representation) [12]. В частности, представление низкого уровня отличается от представления высокого уровня (программы на языке Си) отсутствием структурных управляющих конструкций (`if`, `for` и т. п.).

Для восстановления управляющих конструкций сначала строится граф потока управления программы. По графу потока управления строится дерево доминаторов, затем дуги графа потока управления классифицируются на «прямые», «обратные» и косые.

На основании этой информации уже можно выполнять непосредственно структурный анализ, то есть восстановление высокоуровневых управляющих конструкций [6]. Поиском в глубину в графе выделяются шаблоны основных структурных конструкций, которые затем организуются в иерархическую структуру.

3.4 Восстановление типов

Задача автоматического восстановления типов данных на настоящее время — одна из наименее проработанных с теоретической точки зрения задач в области декомпиляции. Ее можно

условно разделить на подзадачу восстановления базовых типов данных языка, таких как `char`, `unsigned long` и т. п., и на подзадачу восстановления производных типов, таких как типы структур, массивов и указателей. В работе [13] рассматривается восстановление типов, как базовых, так и производных при декомпиляции, однако этот подход имеет ряд существенных недостатков и не имеет практической реализации. В работе [4] описан подход к автоматическому восстановлению производных типов языка по исполняемому файлу. Такой подход используется для анализа на уязвимость программ в виде исполняемых файлов, и поэтому не применим напрямую к задаче восстановления типов при декомпиляции.

На практике же все декомпиляторы, кроме Hex-Rays, вообще не восстанавливают даже базовые типы переменных, а в выражениях используют явное приведение типов, что делает восстановленные выражения сложными для понимания и модификации.

4 Декомпиляторы

В данном разделе дается краткое описание существующих на сегодняшний момент декомпилятров. Это — декомпилятор Boomerang [5], декомпилятор DCC [8], декомпилятор REC [14] и плагин Hex-Rays [10] к дизассемблеру Ida Pro [11]. Все рассматриваемые декомпиляторы, кроме плагина Hex-Rays, на вход принимают исполняемый файл, а на выходе выдают программу на языке Си. В том случае, когда декомпилятор оказывается не в состоянии восстановить некоторый фрагмент исходной программы в язык Си, он сохраняется в виде ассемблерной вставки. Надо заметить, что даже небольшие исходные программы после декомпиляции зачастую содержат очень много ассемблерных вставок, что практически сводит на нет эффект от декомпиляции.

Отличие составляет плагин Hex-Rays, который принимает на вход программу, являющуюся результатом работы дизассемблера Ida Pro, то есть схему программы на ассемблеро-подобном языке программирования. В качестве результата плагин Hex-Rays выдает восстановленную программу в виде схемы на Си-подобном на языке программирования. Тем не менее, для простоты мы в дальшейшем объединим процесс дизассемблирования с использованием Ida Pro и последующей декомпиляции.

4.1 Boomerang

Декомпилятор Boomerang [5] является программным обеспечением с открытым исходным кодом (open source). Разработка этого декомпилятора активно началась 2002 году, но сейчас проект развивается достаточно вяло. Изначально задачей проекта было разработать такой декомпилятор, который восстанавливает исходный код из исполняемых файлов, вне зависимости от того, каким компилятором и с какими опциями исполняемый файл был получен. Для этого в качестве внутреннего представления было решено использовать представление программы со статическими одиночными присваиваниями (SSA). Однако, несмотря на поставленную цель, в результате декомпилятор не сильно адаптирован под различные компиляторы и чувствителен к применению различных опций, в частности, опций оптимизации. Еще одной особенностью, затрудняющей использование декомпилятора Boomerang, является то, что в нем не поддерживается распознавание стандартных функций библиотеки Си.

4.2 DCC

Проект по разработке этого декомпилятора [8] был открыт в 1991 году и закрыт в 1994 году с получением главным разработчиком степени PhD. В качестве входных данных декомпилятор DCC принимает 16-битные исполняемые файлы в формате DOS EXE. Алгоритмы декомпиляции, реализованные в этом декомпиляторе, основаны на теории графов (анализ потока данных и потока управления). Для распознавания библиотечных функций используется сигнатурный поиск, для которого была разработана библиотека сигнатур. Однако надо заметить, что, несмотря на это, декомпилятор плохо справляется с выявлением функций стандартной библиотеки.

4.3 REC

Этот проект [14] был открыт в 1997 году компанией BackerStreet Software, но вскоре закрылся из-за ухода ведущего разработчика проекта. Позднее разработка декомпилятора продолжилась его автором в статусе собственного продукта. Сейчас декомпилятор распространяется свободно, а развивается достаточно вяло. Одной из особенностей рассматриваемого декомпилятора является то, что он восстанавливает исполняемые файлы в различных форматах, в частности ELF и PE. Так же декомпилятор REC можно использовать на различных платформах. В ходе тестирования этого декомпилятора было отмечено, что наиболее успешно декомпилятор восстанавливает исполняемые файлы, полученные при компиляции с включением опций, отвечающих за отключение оптимизаций и добавления отладочной информации.

4.4 Hex-Rays

Как сказано выше, инструмент Hex-Rays [10] не является самостоятельным программным продуктом, а распространяется как плагин к дизассемблеру Ida Pro[11]. Это самое новое из рассматриваемых средств декомпиляции: плагин появился на рынке в 2007 году. Особенностью данного инструмента является то, что он, как уже было отмечено выше, восстанавливает программы, полученные на выходе дизассемблера Ida Pro. В качестве алгоритмов, используемых в рассматриваемом инструменте заслуживают внимания алгоритм сигнатурного поиска FLIRT [1] и алгоритм поиска параметров в стеке PIT (Parameter Identification and Tracking).

В таблице 1 представлена сводная характеристика всех рассматриваемых декомпилятров.

	Boomerang	DCC	REC	Hex-Rays
распознавание библиотечных функций	нет	заявлено	нет	да
активность разработки	да	нет	да	да
переносимость	нет	да	да	да
open source	да	да	да	нет

Таблица 1: Сравнительный анализ существующих декомпиляторов

5 Исследование возможностей декомпиляторов

В этом разделе приведены результаты тестирования возможностей рассмотренных декомпиляторов. Для тестирования был разработан тестовый набор программ на языке Си, покрывающий основные языковые конструкции языка Си.

Тестирование проводилось по следующей методике. Исходный код программы на Си компилировался компилятором gcc 3.4.5 в системе Debian Linux и компилятором Borland C++ 3.1 в системе Windows XP. В первом случае результатом работы компилятора являлся файл формата ELF для архитектуры ia32, во втором случае — исполняемый файл DOS для 16-битного режима процессора. Исполняемый файл формата ELF подавался на вход декомпиляторам Boomerang, REC и Hex-Rays, работающим в системе Windows XP. Исполняемый файл формата DOS EXE подавался на вход декомпилятору DCC. Результат декомпиляции сравнивался с исходным текстом.

Такая комбинация инструментальных и целевых сред была выбрана по следующим причинам. Во-первых, декомпилятор DCC поддерживает только 16-битные исполняемые модули DOS, поэтому для оценки качества работы декомпилятора был использован компилятор 16-битного режима. Декомпиляторы Boomerang и REC наоборот не поддерживают 16-битный режим DOS. Исполняемый модуль подавался на вход декомпиляторам в формате ELF, а не в естественном для Windows формате PE, так как декомпиляторы Boomerang и REC, как оказалось, некорректно обнаруживали точку начала программы на Си в файлах формата PE.

Количество баллов за тест	Комментарий
0	Декомпилятор закончил работу с ошибкой выполнения или пустым результатом.
1	Декомпилятор выдал ассемблерный код. Программа на Си не была получена.
2	Декомпилятор выдал программу на языке Си, которая либо не компилируется, либо работает неверно (неэквивалентна исходной), либо содержит ассемблерные вставки, то есть недекомпилированные фрагменты программы.
3	Декомпилятор выдал корректную программу, которая эквивалентна исходной, но использует конструкции, отличные от конструкций в исходной программе.
4	Декомпилятор выдал программу, которая эквивалентна исходной и использует те же конструкции, которые использовались в исходной программе.

Таблица 2: Шкала оценки декомпиляторов

Качество работы каждого декомпилятора для каждого теста оценивалось по 4-бальной экспертной шкале, приведенной в таблице 2. Так, оценка «3» выставлялась в случаях, когда декомпилированная программа использовала адресную арифметику вместо массивов или приведения типов для получения указательных значений вместо корректного объявления типов переменных. Кроме того, оценка «3» выставлялась, если в результате декомпиляции цикл `for` оказывался преобразовыванным в цикл `while`.

5.1 Система тестов

Тестовый набор содержал следующие основные группы.

- **Типы.** В тестах этой группы проверялась корректность восстановления типов переменных и параметров функций. Язык Си поддерживает богатый набор базовых целочисленных типов от типа `char` до типа `unsigned long long`. Декомпилятор должен по возможности точно восстановить как размер, так и знаковость переменной.

  Также рассматривались типы указателей на базовые типы. Проверялся факт обнаружения того, что переменная является указательным, а не целым типом, и корректность восстановления целевого типа указателя.

  Для массивов проверялся факт обнаружения того, что переменная является локальным или глобальным массивом, точность восстановления типа элементов массива, точность восстановления размера массива как для одномерных, так и для многомерных массивов.

  Для структурных типов проверялся факт распознавания использования структурного типа и точность восстановления полей структур.

  Кроме того, были рассмотрены разные комбинации указательных, массивовых и структурных типов и оценена корректность восстановления таких составных типов. В частности, рассматривались массивы структур, указатели на структуры, структуры, содержащие массивы, структуры, содержащие указатели на самих себя.

- **Языковые конструкции.** В тестах этой группы проверялась корректность восстановления управляющих структур программы. Проверялась корректность восстановления оператора `if` с простым условием, в том числе и с отсутствующей частью `else`, операторов цикла `while` и `do while` с простыми условиями.

  В другой группе тестов проверялась корректность восстановления логических операций `&&` (логическое «и»), `||` (логическое «или») в условиях операторов `if` и циклов. Согласно семантике языка Си эти операторы транслируются в условные и безусловные переходы,

то есть являются конструкциями, задающими поток управления, а не вычисления значений. Декомпиляторы должны по возможности восстанавливать сложные условия в операторах языка.

Отдельно проверялась корректность восстановления структурных операторов передачи управления, таких как `break`, `continue` и `return`.

Оператор `switch` рассматривался отдельно, так как в большинстве компиляторов он транслируется в косвенный безусловный переход, где адрес перехода выбирается из таблицы в соответствии с вычисленным в заголовке оператора значением. Декомпиляторы должны разпознавать использование этого оператора в программе.

- **Функции.** В тестах этой группы проверялась корректность выделения параметров функций и локальных переменных в условиях разных соглашений о вызовах. Кроме того, проверялась корректность обработки рекурсивных функций.

- **Оптимизации.** В тестах этой группы проверялась корректность работы декомпиляторов при условии, что при компиляции были использованы некоторые оптимизационные преобразования, такие как открытая вставка функций (inlining) и оптимизации вызовов функций (tail call optimization, tail recursion optimization, sibling call optimization).

- **Взаимодействие с окружением.** В данной группе находился тест, проверяющий корректность обнаружения функции `main` в исполняемых файлах формата PE. Как известно, выполнение программы на языке Си начинается с функции `main`, которой передается определенный список параметров. Однако в исполняемых файлах вызову функции `main` предшествует выполнение специального кода, задача которого настроить окружение программы на Си, что заключается, в частности, в создании стандартных потоков ввода-вывода, инициализации служебных структур данных управления динамической памятью и т. п. Этот код частично написан на языке ассембера, кроме того, он не представляет интереса, так как стандартный для всех программ. Поэтому декомпиляторы должны игнорировать этот инициализационный код и начинать декомпиляцию непосредственно с функции `main`.

  Кроме того, в тестах этой группы проверялось распознавание стандартных библиотечных функций языка Си (например, `strlen` и т. п.). Реализация сигнатурного поиска присутсвует в декомпиляторе DCC, но наиболее развита эта технология в декомпиляторе Hex-Rays.

5.2 Результаты тестирования

В таблице 3 приводятся результаты работы декомпиляторов на выбранном наборе тестов в соответствии с системой оценок, указнной в таблице 2. Каждый столбец таблицы соответствует декомпилятору, а каждая строка — тесту. Общий результат для каждого декомпилятора получен суммированием оценок по всем тестам.

Из всех рассмотренных декомпиляторов только Boomerang поддерживает декомпиляцию оператора `switch`. Остальные декомпиляторы генерируют в этом случае некорректный код на языке ассемблера. Только декомпилятор REC сумел восстановить цикл `for`, в то время как остальные декомпиляторы в этом случае генерируют программу, использующую цикл `while`.

Наиболее развитым в настоящее время является декомпилятор Hex-Rays, который, в отличие от других декомпиляторов, поддерживает распознавание массивов и распознавание библиотечных функций, хотя даже и Hex-Rays имеет много слабых сторон.

6 Заключение

В данной работе рассмотрена задача декомпиляции как восстановления программы на языке высокого уровня по программе на языке ассемблера или в машинных кодах. Дано краткое описание основных шагов процесса декомпиляции программы. В работе представлено сравнительное тестирование существующих декомпиляторов и указаны их сильные и слабые стороны.

Декомпилятор/ Тестовый пример	Boomerang	REC	DCC	Hex-Rays
Типы: `struct`	3	2	3	3
Типы: массивы	4	3	3	4
Типы: встроенные, `unsigned int`	4	3	3	3
Типы: встроенные, `unsigned short`	3	3	3	3
Структуры: `if`	4	4	4	4
Структуры: логические операции	4	4	4	4
Структуры: циклы `for`	3	4	3	3
Структуры: циклы `while`	4	3	4	4
Структуры: циклы `do while`	4	4	4	4
Структуры: `switch`	4	2	2	2
Функция: рекурсия	4	4	4	4
Оптимизации: inlining	2	2	—	2
Оптимизации: tail recursion	2	2	—	2
Обнаружение main в PE файлах	1	1	—	4
Обнаружение стандартных функций	2	2	3	4
Сумма баллов:	48	43	40	50

Таблица 3: Результаты тестирования декомпиляторов

Так, ни один существующий на данный момент декомпилятор не поддерживает в достаточной мере восстановление типов данных, как базовых так и производных. Существующие декомпиляторы испытывают сложности с восстановлением цикла `for` и оператора `switch`. Наиболее развитым из всех декомпиляторов является Hex-Rays, однако он является коммерческим и с закрытыми исходными кодами, поэтому его доработка невозможна.

Поэтому представляются актуальными разработка и реализация алгоритмов, позволяющих восстанавливать базовые и производные типы данных в процессе декомпиляции. Алгоритмы должны быть апробированы в рамках экспериментальной инструментальной среды для декомпиляции программ. Разработка таких алгоритмов и инструментальной среды декомпиляции программ является направлением дальнейших исследований авторов.

Список литературы

[1] Гуильфанов И. FLIRT — Fast Library Identification and Recognition Technology. http://www.idapro.ru/description/flirt/

[2] Щеглов К.Е. Обзор алгоритмов декомпиляции//Электронный журнал «Исследовано в России». http://zhurnal.ape.relarn.ru/articles/2001/116.pdf

[3] AT&T Assembly Syntax. http://sig9.com/articles/att-syntax

[4] G. Balakrishnan, T. Reps. Analyzing Memory Accesses in x86 Executables. Compiler Construction vol. 2985/2004, Springer Berlin / Heidelberg, 2004, стр. 5-23.

[5] Boomerang Decompiler Home Page. http://boomerang.sourceforge.net/

[6] C. Cifuentes, D. Simon, A. Fraboulet. Assembly to High-Level Language Translation. Technical Report 439. Department of Computer Science and Electrical Engineering. The University of Queensland. 1998.

[7] M. Davis. Computability and Unsolvability, New York: McGraw-Hill, 1958.

[8] DCC Decompiler Home Page. http://www.itee.uq.edu.au/~cristina/dcc.html

[9] Agner Fog. Calling conventions for different C++ compilers and operating systems.

[10] Hex-Rays Decompiler SDK. http://www.hex-rays.com/

[11] Интерактивный дизассемблер Ida Pro. http://www.idapro.ru/

[12] Steven S. Muchnik. Advanced Compiler Design And Implementation.

[13] A. Mycroft. Type-based decompilation. In European Symp. on Programming, 1999.

[14] REC Decompiler Home Page. http://www.backerstreet.com/rec/

[15] Tool Interface Standards (TIS). Executable and Linkable Format (ELF). http://www.x86.org/intel.doc/tools.htm

[16] Tool Interface Standards (TIS). Portable Executable Formats (PE). http://www.x86.org/intel.doc/tools.htm

УДК 004.855.6

АДАПТИВНЫЙ ПОИСК В ИНТЕЛЛЕКТУАЛЬНОЙ ОБУЧАЮЩЕЙ СИСТЕМЕ

© 2008 г. А. А. Фролов

afrolov@its-grom.org

Кафедра Алгоритмических языков

1 Введение

Проектирование и построение интеллектуальных обучающих систем (ИОС) уже дважды меняло свою базовую концепцию в мировом историческом развитии. В первый раз это означало переход от основных идей искусственного интеллекта, а именно экспертных диагностических и классификационных систем, к гипертекстовым системам, предоставившим теоретически неограниченные возможности навигации по материалу. Во второй — переход к системам, поддерживающим определенные психологические теории, такие как «теория множественного интеллекта» Говарда Гарднера [1] или «герменевтики развивающего обучения метод» В. И. Громыко [2]. В настоящий момент происходит переход к распределенной многомодульной архитектуре обучающих систем, позволяющей оперировать со значительным количеством источников знания. Это позволяет привлекать к участию в разработке ИОС очень широкий круг людей, а также переформулировать задачу обучения в задачу адаптивного поиска по этим источникам.

Несмотря на то, что основная задача создания автоматизированного учителя с течением времени не изменилась, усложнился ее контекст и формализовалось понимание. В настоящее время для успешной реализации ИОС необходимо решить ряд задач:

- Универсальность системы относительно источников знания: обеспечение работы со множеством существующих учебных материалов. Огромное количество научного, учебного и профессионального материала полностью компенсируется разнообразием форм, в которых он существует. В рамках эксперимента, проведенного на семинаре, установлено, что для грубой обработки и разметки учебника, даже при наличии полу-автоматизированных программных средств, требуется около суток. В настоящий момент параллельно происходит построение как интерфейсов для преобразования информации, так и средств преобразования во внутреннее представление.

- Удобное представление материала: графы, элементы презентаций, учебные модули и динамически формируемые части учебных курсов. Система должна обеспечивать удобную работу с максимальным количеством разнообразных видов обобщенного представления информации, включая синтезированные.

- Адаптивное представление материала: поиск материала, соответствующего пониманию решения задачи учащимся в системе обучения. В связи с резким увеличением числа источников знания (научных статей, книг), в том числе и в электронном виде, эта задача становится ключевым местом общей концепции.

- Модель обучения: общая стратегия развития системы (например, поиск наиболее подходящих видов интеллекта в рамках модели множественного интеллекта) и соответствующая организация подачи материала. Выбор и формализация модели зависит от целей реализации системы.

- Модель учащегося: история взаимодействия с учащимся, изменение некоторой априорной модели (например, байесовской сети), структуры, синтезированные во время работы с системой.

- Открытость, доступность и масштабируемость: обычно проекты реализации носят краткосрочный характер, поэтому отсутствие стандартов в архитектуре приводит к снижению их продуктивности.

- Способность интегрировать знания большого числа людей: П. Брусиловский в своей статье [3] пишет о необходимости предоставления учителям возможности принимать участие в формировании если не стратегии обучения системы, то хотя бы учебного материала.

Настоящая статья посвящена детальной разработке и начальной реализации поискового алгоритма, представляющего собой фактически первый эксперимент по реализации комплекса программных средств помощи учащемуся. Разработанная реализация называется ИОС Сатори. Работа поискового алгоритма рассматривается только в рамках обучения информатике студентов 1-го и 2-го куров математических отделений ВУЗов. В качестве предметной области рассматриваются исключительно курсы по информатике или же связанные с ними курсы по другим естественно-научным дисциплинам. Используется модель обучения, разработанная В. И. Громыко [2].

2 Механизм интеллектуального поиска

2.1 Поиск в ИОС

ИОС обеспечивает удобную работу с материалом и позволяет осуществлять интеллектуальный поиск по нему. Предметная область ограничена информатикой, что позволяет использовать ее специфику, поэтому в качестве базовой концепции используется поиск по метаданным.

Проблема большого количества материала решается за счет адаптивной настройки на учащегося, работы с материалом на уровне не ниже понятийного и механизма визуализации. За счет экспертного знания и развития учащегося на предметной области, в которой он работает, удается преодолеть неточные запросы самого учащегося. Ориентированность поиска на ключевые проблемы предмета позволяет осуществлять работу даже при нестрогом определении цели поиска. Неточность запроса связана с тем, что учащийся, работая в новой для себя области, не может достаточно хорошо в ней ориентироваться.

2.2 Адаптивный поиск

Задача поиска естественным образом ставится уже самим учащимся во время формулирования некоторой интеллектуальной проблемы по овладению знанием конкретного учебного материала. Определенное представление об устройстве механизма поиска заранее заложено в систему в качестве набора стратегических и тактических решений. Разработанный механизм адаптивного поиска служит удовлетворению запросов учащегося в рамках общей модели его развития.

Необходимость и актуальность поисковых методов работы являются следствием необъятности учебного материала и невозможности построения универсальной функции, то есть некоторого алгоритма, который на любой предметной области позволял бы точно моделировать мышление учащегося. Невозможность работы одновременно с десятками учебных курсов обычно приводит к простому узнаванию методов решения конкретных задач, что не вполне соответствует целям, которые хотел поставить перед учащимся автор учебного курса.

С точки зрения учащегося этот механизм управления и представления связности материала является методом концептуального исследования неизвестных ему ранее предметных областей.

В рамках задачи поиска вводятся термины стратегия и тактика. Стратегия является долговременной реализацией развития учащегося на пути перехода по определенным интеллектуальным состояниям [2], или интеллектуальным прорывом. Она обеспечивается формализацией в методе ГРОМ представления о направлении развития учащегося в соответствии с состоянием в его динамической модели. С точки зрения искусственного интеллекта она представляет собой вид метазнаний в понимании Д. Поспелова [4]. Тактика — это множество методов, реализуемых в рамках стратегии для достижения определенного промежуточного результата.

Каждая тактика отвечает за целесообразность выбора определенного маршрута на некотором множестве материала, актуального в контексте текущей стратегии. В отличие от стратегии, реализация которой может потребовать значительного времени, тактика направлена на исправление конкретных недостатков учащегося. Конечной целью каждой тактики является борьба с определенным формальным проявлением «болезни учащегося» в рамках общей модели развития учащегося в системе.

Результатом поиска являются:

- Соответствующие ключевые задачи в найденных курсах называемые конкретными пример-проблемами.
- Понятия, образующие эти пример-проблемы.
- Некоторое множество учебных элементов текста, базовых частей, на которые был разделен курс в момент преобразования во внутреннее представление.

Поиск представляет собой итеративный процесс, в результате которого, в частности, может осуществляться маршрутизация по материалу вверх или вниз [5]. Маршрутизация отличается от обычной навигации динамическим и адаптивным формированием материала.

3 Архитектура поиска в ИОС Сатори

Архитектура ИОС Сатори представляет собой трехуровневую модель. Каждый из уровней содержит в себе определенное представление материала. Сама система является своего рода интерфейсом для доступа к материалу. Учащийся, стремясь получить интересующий его учебный материал, использует функциональность адаптивного уровня, представленную механизмами поиска, кластеризации и фильтрации. Они, в свою очередь, взаимодействуют с моделями материала и учащегося, реализующими уровень представления.

Уровень источников знаний образуют задачники и учебники различных типов и уровней. В него входят различные авторские курсы, покрывающие различные части модели предметной области в рамках метода обучения. Курсы подразделяются на абстрактные, детальные и задачники. Абстрактные курсы форимруют модель предметной области, а детальные и задачники более подробно описывают некоторую ее часть и содержат большое количество примеров. В рамках проводимого эксперимента, примером абстрактного курса можно считать курс Э. Энгелера «Метаматематика элементарной математики», примером детального — учебник В. А. Ильина и Э. Г. Позняка «Основы математического анализа: В 2-х ч. Часть 2». Кроме того, на этом уровне могут находится любые другие источники, например, учебные модули других интеллектуальных систем.

Уровень представления содержит в себе структуры, синтезированные системой и учащимся на основании уровня источников знаний и работы учащегося в системе. Структуры представляют собой ранжированные графы, демонстрирующие связность материала из смежных предметных областей, внутреннее устройство материала и модель учащегося, как его проекцию на учебный материал.

Уровень адаптации представлен структурами, сгенерированными по запросу учащегося и отвечающими его модели в системе, которая в свою очередь соответствует цели его работы. В рамках этого уровня в системе имеются механизмы кластеризации, фильтрации, осуществления запросов к системе и автоматической генерации адаптивных структур.

4 Модель поиска в рамках реализации ИОС Сатори

Граф элементов курса содержит элементы курса и связи между ними. Позволяет организовывать материал различными способами. Находится в целостном состоянии с соответственным графом понятий.

Граф понятий курса содержит понятия курса, связи между ними и связи с понятиями других курсов.

Карта курса содержит конкретные пример-проблемы. Пример-проблема включается в граф в том и только том случае, если пересечение множества понятий, образующих данную пример-проблему, и множества понятий, образующих динамическую модель учащегося, составляет более половины понятий пример-проблемы. Веса, приписываемые ребрам в графе, соответствуют количеству общих понятий двух пример-проблем.

Граф модели учащегося строится на основе понятий, составляющих модель учащегося.

4.1 Начало работы: выбор учебного курса и пример-проблемы

Учащийся, работая с соответствующими визуальными представлениями, выбирает учебный курс, исходя из своих целей обучения или своей оценки знания материала. У него есть возможность просматривать все конструкции, поддерживающие курс. Выбрав курс, учащийся получает доступ к новой структуре, содержащей все пример-проблемы, образующие этот курс. И, в свою очередь, снова делает выбор наиболее приемлемой пример-проблемы. Возможен вариант выбора определенного понятия, в таком случае пример-проблема будет подобрана автоматически.

Курс зафиксирован в некоторой точке метамодели предметной области, а также в некоторой иерархии, развернутой внутри точки метамодели. Положение курса в рамках метамодели предметной области позволяет сделать вывод о проблемах, которые возникают на пути интеллектуального развития учащегося.

4.2 Анализ состояния учащегося: выбор тактики и поиск материала

1. Происходит выбор тактики. Тактика задает поведение системы, определяемое правильным представлением о развитии учащегося, исходя из выбранного курса.

   Примеры:

   - Направление обучения в сторону «школьного уровня». Соответствует ситуации, когда выбранный курс находится вне множества верхних точек иерархии курсов. Система делает вывод, что в этом направлении учащегося развивать легче в силу того, что он уже имеет какое-то представление об устройстве предмета. Предлагаемые пример-проблемы будут в первую очередь браться из той же точки модели предметной области, но ближе к вершине иерархии.
   - Направление обучения в сторону «вузовского уровня». Аналогично предыдущему случаю, система делает вывод, что учащемуся легче развиваться через множественные системы аксиом, через программирование, так как именно в этой области у него в настоящий момент существует достаточно большой опыт.
   - Направление обучения в сторону «вузовского уровня» от «школьного уровня». В случае, если выбранный курс вполне соответствует «школьному уровню», то есть является фундаментальным курсом области, находящимся в вершине иерархии, или если в рамках работы в системе модель учащегося сама соответствует такому курсу, принимается решение о поиске пример-проблем в смежных областях (но по-прежнему согласованных с моделью учащегося). В случае «школьного уровня» в соответствии с методом обучения будет выдвинута гипотеза, согласно которой обучающийся имеет определенные проблемы с работой со сложными и разнообразными моделями, что связано с отсутствием у него опыта в данной области. Этот факт будет иметь решающее значение при выборе пример-проблем.
   - Направление обучения в сторону «школьного уровня» от «вузовского уровня». Общая концепция поведения аналогична предыдущей. В соответствии с методом обучения, учащийся, находящийся в точке, относящейся к «вузовскому уровню», недостаточно хорошо понимает реальность, связанную с теми моделями, с которыми ему приходится иметь дело. Другими словами — не хватает хороших конкретных примеров. Тактика, реализуя метод обучения, будет направлена на решение именно этой проблемы.

2. Осуществляется поиск более конкретных пример-проблем в соответствии с предыдущим шагом.

3. Осуществляется поиск элементов материала, который поддерживает выбранные пример-проблемы.

4.3 Изменение структур: настройка связей

На предыдущем этапе учащийся получает из различных точек модели предметной области большое количество разнообразного материала и пример-проблему. Результат его может как устроить, так и не устроить. В любом случае в рамках системы ему предоставляется доступ к графам, поддерживающим курс и его собственную модель. Он может добавлять или удалять новые понятия, выстраивать между ними направленные и ненаправленные связи. Он может:

- Изменять состав учебного курса в соответствии со своими представлениями. То есть отсекать элементы курса, помечая их как непонятные.
- Расширять свою динамическую модель, добавляя туда новые понятия.
- В дальнейшем осуществлять настройку одного курса на другой, выстраивая аналогии и другие связи между понятиями.

Его целью является построение структур, грубо соответствующих его настоящему интеллектуальному состоянию. В получившихся графах не должно остаться понятий, которые, с точки зрения учащегося, совершенно непонятны либо же бесполезны. После осуществления данной работы учащийся может запросить систему провести повторный поиск.

4.4 Макро-операции: коррекция системы

Предусмотрен ряд макро-операций, предназначенных для сильной корректировки работы системы. Это может понадобиться в том случае, если система выдает совершенно неадекватные материалы с точки зрения учащегося. Рассматриваются следующие операции:

- классификация материала как слишком сложного;
- классификация материала как слишком легкого;
- классификация материала как нерелевантного запросу.

Информация, полученная системой в результате применения этих операций, несет критически важное значение для поиска и учитывается при каждой следующей итерации работы системы. Релевантность «неподходящего» материала понижается, выбираются другие пример-проблемы и другой материал для обучения.

5 Архитектура ИОС Сатори

В качестве основной цели ставилось получение реализации, в первую очередь направленной на осуществление поддержки курсов, в том числе и для создания уровня представления. Для этого были использованы программные средства, поддерживающие быстрый цикл программирования. В качестве языков были выбраны Ruby [8] и интегрированный в его библиотеку язык Tk. Ruby хорошо приспособлен для таких проблемных областей, как обработка текста и XML.

Система имеет модульную структуру и состоит из следующих частей:

- *Модуль управления* отвечает за синхронизацию работы остальных модулей системы и передачу сообщений.

- *Модуль по работе с графами* включает в себя классы, реализующие работу с графами. Содержит как базовые средства для работы с графами, так и определенную логику, определяющую каждый конкретный тип графов.
- *Модуль поиска* включает в себя логику поиска материала на основании метода ГРОМ и средства его обобщения.
- *Модуль работы с XML объектами* отвечает за работу с файлами модели учащегося, учебных курсов и общего индекса системы.
- *Модуль пользовательского интерфейса* реализует графический оконный интерфейс.

ИОС Сатори является экспериментальной реализацией и не осуществляет полный цикл работы интеллектуальной обучающей системы. В частности, исходные учебные курсы принимаются только в определенном описанном XML формате. В настоящее время XML файлы являются основным способом передачи данных между ИОС Сатори и другими частями разрабатываемой на семинаре ИОС.

Для интеграции различных курсов в рамках существующей архитектуры приходится настраивать связи вручную. При этом уже разработан начальный вариант общего индекса, так называемый универсальный индекс понятий, предназначенный для централизованного хранения метаданных об устройстве понятий в используемых авторских курсах и задачниках. Он позволяет устанавливать отношения эквивалентности и иерархии между понятиями одного или нескольких курсов.

6 Реализация

Основной проблемой, решаемой в связи с визуализацией структур ИОС, является обеспечение удобной работы с очень большим количеством объектов. В частности, используемые обработанные курсы содержат:

Название курса	Элементы текста	Понятия
«Метаматематика элементарной математики»	672	94
«Основные понятия алгебры»	1335	499
«Лекции по математике: анализ. Т.1»	1051	199
«Линейная алгебра»	2229	381
«Общая алгебра»	511	175
«Универсальная алгебра»	1747	813
«Алгебра»	4554	1578

Таблица 1: Количество объектов курсов

Таким образом, уже при работе с 3–5 курсами, общее количество объектов графов, содержащих понятия и элементы курса, достигает нескольких тысяч, в то время как для эффективной работы учащегося с разработанными инструментами на графической канве не должно быть более 10–50 объектов.

Для визуализации были использованы визуальные схемы, реализующие в себе средства работы с графами. Они являются интерфейсом между учащимся и системой. С их помощью учащийся или человек, создающий предметную область, могут осуществлять работу по исследованию материала, выстраиванию в нем связей, поиску.

6.1 Модель учащегося

Работа с компонентами системы, входящими в модель учащегося, реализована при помощи двух визуализационных схем: графа понятий модели учащегося и графа метамодели.

Граф понятий модели учащегося позволяет добавлять понятия из индекса учебных курсов и выстраивать между ними направленные связи. Связь между понятиями интерпретируется как обобщение или ассоциация.

Граф метамодели содержит загруженные в модель учащегося курсы. Поддерживается возможность построения связей между ними, которые интерпретируются системой как моделирующие локальную иерархию в рамках определенной точки метамодели. Чем выше по этим связям находится курс, тем он менее конкретен. То есть самыми нижними вершинами должны являться задачники и учебники, подробно описывающие конкретную область знания.

6.2 Авторские курсы

Главное окно программы реализует интерфейс доступа к учебному курсу, позволяя осуществлять просмотр индекса понятий, элементов текста и понятий индекса, ассоциированных с ними. Предусмотрена возможность перехода на определенный элемент текста. Кроме этого, есть возможность изменять положение учебного курса в рамках метамодели предметной области. Главная задача, реализуемая в рамках данной функциональности, — обеспечение работы с понятием.

Основной схемой является граф понятий курса. Он позволяет выстраивать отношения между понятиями в рамках как одного авторского курса, так и определенного множества курсов. Связи могут быть следующих типов:

- эквивалентность (ненаправленная);
- авторская иерархическая связь (направленная);
- представление учащегося (направленная).

Для реализации операции, обеспечивающий представление связности материала, были разработаны автоматические генераторы внутренней структуры:

- Авторская структура: добавляет связь между понятиями в случае, если в курсе существует элемент текста к которому относятся оба понятия.
- Кластеризация: из авторской структуры удаляются понятия с очень большим количеством связей. Из оставшихся вершин при помощи метода иерархической кластеризации выделяется N кластеров.
- Связь с моделью учащегося: оставляет лишь те понятия, которые входят в модель учащегося или напрямую связаны с ними.

Для уменьшения количества объектов был построен механизм фильтрации. Фильтрация может осуществляться по минимальному количеству связей в графе и шаблону текста.

Кроме этого существуют карта курса и схема с графом элементов текста, позволяющие моделировать связи на множествах пример-проблем и элементов текста соответственно.

Граф элементов позволяет работать с физической структурой курса. Он находится в целостном состоянии относительно графа понятий. В нем существуют связи двух типов:

- авторская иерархическая связь (направленная);
- представление учащегося (направленная).

Карта курса показывает курс или модель учащегося с точки зрения образующих их пример-проблем. Связи соответствуют входящим в пример-проблемы понятиям.

6.3 Поиск

Поиск реализован в качестве функциональности окна главной программы. Для того, чтобы выполнить поисковый запрос к системе пользователь, в соответствии с алгоритмом, должен:

1. выбрать абстрактную-пример проблему или курс;

2. отредактировать граф понятий модели учащегося;

3. выполнить запрос.

После этого система может сформировать для него результаты запроса, сформулированные в виде нового «курса», который можно увидеть в списке используемых курсов.

Механизм поиска позволяет увеличить эффективность нахождения учебного материала для неформализованных целей учащегося.

7 Перспективы

В процессе эволюции обучающей системы количество макро-операций должно увеличиваться. Тонкая ручная настройка материала для выполнения эффективного поиска показала свою неэффективность в рамках работы с очень большим количеством объектов. В ходе тестирования на пяти различных авторских курсах было установлено, что количество элементов текста одного курса ограничено сверху 2000–3000, а количество понятий составляет 200–500.

Механизм поиска должен обрабатывать максимально возможное количество информации о студенте, его использующем. Возможным решением является интеграция концепции атомарных байесовских сетей [6] в модель учащегося.

Хорошим архитектурным решением автору видится использование распределенной среды для хранения документов, где в качестве транспорта использовался бы уже ставший достаточно популярным (его поддерживают около десяти систем) в разработке систем e-learning интерфейс SQI [7].

Кроме того, необходимо понизить детальность описания исходных учебных данных, отделив от них метаданные об их устройстве. Это позволит использовать более широкий круг источников знаний, а также повысит переносимость и интегрируемость разрабатываемого решения.

Список литературы

[1] Гарднер Г. *Структура разума. Теория множественного интеллекта.* Вильямс, 2007

[2] Громыко В. И. *Искусство рационального.* Синергетика. Труды семинара, том 8. М.: МГУ, 2006

[3] Peter Brusilovsky *Supporting teachers as content authors in intelligent educational systems* Int. J. Knowledge and Learning, Vol. 2, Nos. 3/4, 2006 191

[4] Поспелов Д. А. *Десять «горячих точек» в исследованиях по искусственному интеллекту.* Интеллектуальные системы (МГУ). - 1996. - Т.1, вып.1-4. - С.47-56.

[5] Громыко В. И., Аносов С., Кондаков А., Крылов С., Фролов А. *Интеллектуальные обучающие системы для базового обучения информатике (реализация).* Актуальные проблемы информатики в современном российском образовании. 2004

[6] Wei F., Blank G. D. *Student Modeling with Atomic Bayesian Networks.* Lehigh University. 8th International Conference ITS, 2006

[7] Nhu Van Nguyen and David Massart *Binding the Simple Query Interface* European Schoolnet, rue de Trives 61, B-1040 Brussels, Belgium, 2007

[8] Carlson L., Richardson L. *Ruby Cookbook.* O'Reilly Media, 2006.

УДК 004.432

БИБЛИОТЕЧНАЯ РЕАЛИЗАЦИЯ ВЫЧИСЛИТЕЛЬНОЙ МОДЕЛИ ЯЗЫКА ПЛЭНЕР

© 2008 г. О. Г. Фролова

intafy@gmail.com

Кафедра Алгоритмических языков

1 Введение

В настоящее время для проектирования и разработки крупных программных проектов в большинстве случаев используются индустриальные языки программирования, которые, как правило, являются объектно-ориентированными. Такими языками, к примеру, являются C++, C#, Java.

В то же время при разработке подобных проектов могут возникать разнообразные подзадачи, для которых применение традиционных языков неэффективно с точки зрения использования труда программиста. Примерами таких задач могут служить задачи искусственного интеллекта, компьютерной лингвистики, задачи, связанные с преобразованиями деревьев, графов, математических формул и т. п. При этом они могут быть достаточно хорошо решены средствами языков функционального или логического программирования. Но эти языки, в свою очередь, либо менее удобны для задач описания графических интерфейсов, взаимодействия с операционной системой, либо оказываются недостаточно эффективными по времени работы или расходу памяти. Одним из возможных подходов к разработке сложных программных систем является одновременное использование различных языковых средств и различных парадигм программирования.

Planner является одним из наиболее известных языков программирования для искусственного интеллекта. Этот язык был разработан Карлом Хьюиттом в Лаборатории искуственного интеллекта Массачусетского Технологического Института, первые публикации о нем появились в 1969 г. [1]

Этот язык включает в себя в качестве подмножества модификацию языка Лисп, расширяя его возможностями анализа данных по образцам, поиска с возвратами, работы со встроенной базой данных, дедуктивным механизмом.

Таким образом, Planner ориентирован на задачи обработки символьной информации, решаемые методами перебора и дедуктивных выводов. Язык является удобным средством для решения задач искусственного интеллекта, таких, как планирование действий робота, доказательство теорем, вопросно-ответные системы, понимание естественного языка.

Для реализации был выбран диалект, разработанный на кафедре Алгоритмических языков факультета ВМК МГУ и получивший название Плэнер [7], [9]. Он представляет собой некоторое упрощение оригинального варианта языка. В частности, в нем отсутствуют параллельные процессы, изменен синтаксис ряда встроенных процедур, ограничено использование сопоставителей — процедур, осуществляющих расширенное сопоставление с образцом. Такие изменения упрощают язык и позволяют его эффективнее реализовывать и использовать, но практически не сужают область применения языка.

2 Метод непосредственной интеграции и библиотека InteLib

Объединение различных стилей программирования возможно, например, с помощью метода непосредственной интеграции [3]. Согласно этому методу в качестве основного средства разработки проекта выбирается объектно-ориентированный язык программирования, предоставляющий возможность переопределения символов арифметических операций. Семантики требуемых альтернативных языков рассматриваются как специфичные предметные области,

которые моделируются средствами основного языка в виде объектно-ориентированных библиотек классов. При этом определяются специальные пользовательские типы и некоторые инфиксные операции над ними, что позволяет промоделировать изобразительные средства вспомогательных языков в виде арифметических выражений базового.

Таким образом, поддержка функциональности вспомогательного языка реализуется библиотекой классов, которую можно использовать для написания частей большого программного продукта. Код, написанный с использованием такой библиотеки, получается синтаксически похожим и семантически эквивалентным коду на вспомогательном встраиваемом языке, оставаясь при этом программой на основном языке разработки проекта.

Метод непосредственной интеграции реализован в проекте InteLib [6], [4]. В качестве основного языка в библиотеке InteLib выбран язык C++. В настоящее время библиотека предоставляет поддержку языков Lisp, Scheme, Refal. Библиотека реализует S-выражения [10] как гетерогенные структуры данных [5] и развитые средства работы с ними.

Базовым классом библиотеки является абстрактный класс `SExpression`. На его основе построена иерархия наследования, моделирующая S-выражения разных типов, например, точечные пары (класс `SExpessionCons`), целочисленные константы (класс `SExpressionInt`), числа с плавающей точкой (класс `SExpressionFloat`), символы-метки (класс `SExpressionLabel`) и некоторые другие. В библиотеке реализованы «умные» указатели на S-выражения (класс `SReference`) и метод счетчика ссылок для сборки мусора.

Для реализации вычислительной модели языка Плэнер в рамках библиотеки InteLib были разработаны структуры данных для представления конструкций языка на основе S-выражений и с их помощью реализованы основные возможности языка.

3 Особенности реализации

3.1 Структуры данных

В Плэнере, как и в Лиспе, для изображения программ и данных используется единственный вид конструкций: выражения, которые делятся на атомарные и списковые. В отличие от Лиспа в Плэнере используются три типа списков. Синтаксически они отличаются по виду скобок, которыми ограничены элементы. Списки с круглыми скобками принято называть L-списками, списки с квадратными скобками — P-списками, списки с угловыми скобками — S-списками.

Поскольку синтаксис L-списков в Плэнере совпадает с синтаксисом списков языка Лисп, для представления L-списков используется имеющийся в библиотеке InteLib класс `SExpressionCons`. Для представления двух других типов списков вводятся дополнительные структуры. P-списки представляются классом `PlnExpressionCons`. Вычисление S-списков отличается от P-списков только тем, что результат вычисления должен быть при дальнейшем его использовании сегментирован. Аналогичным свойством обладают и сегментные обращения к переменным. Поэтому для представления всех сегментированных выражений было решено использовать объекты одного общего класса `PlnExpressionSegmented`. Он содержит внутри себя «умный» указатель на простую форму, соответствующую представляемой сегментной форме.

В основе библиотеки InteLib, кроме классов иерархии S-выражений, лежит класс `SReference` — «умный» указатель на S-выражения. От него был унаследован класс `PlnReference` — «умный» указатель на S-выражения, соответствующие объектам Плэнера, имеющий специальные методы для вычисления плэнерских форм и проверки списков различных типов на пустоту. В нем перегружен оператор «,», который теперь используется для создания списков: левый операнд должен быть списком одного из трех типов, к этому списку в хвост дописывается новый элемент — правый операнд. При этом действие оператора «,» различается в зависимости от типа левой части, чтобы получить корректные плэнерские списки. Таким образом, если объект `list` класса `PlnReference` является «умным» указателем на объект класса `PlnExpressionCons`, соответствущий некоторому списку в квадратных скобках, то результатом вычисления выражения языка C++ `(list, 5)` станет снова «умный» указатель на список в квадратных скобках.

Для более удобного конструирования списков были введены два новых класса: `PlnLListConstructor` и `PlnPListConstructor`. Для них перегружен оператор «|», который

возвращает список в круглых или квадратных скобках из одного элемента — своего правого операнда. Кроме того, в классе `PlnReference` перегружен оператор «!», сегментирующий его.

В Плэнере идентификаторы вычисляются сами в себя, а для обращения к локальным и глобальным переменным используются шесть типов префиксов. Для представления идентификаторов был создан класс `PlnExpressionLabel`, объекты которого хранят в себе имя идентификатора и значение соответствующей константы или функции. Эти же объекты связываются со значениями локальных переменных в лексическом контексте. Классы для представления обращений к переменным являются «обертками» `PlnExpressionLabel` и различаются только способом вычисления.

Класс `PlnSymbol` является «умным» указателем на `PlnExpressionLabel` и был введен ради перегружаемых в нем операций. Сам по себе объект этого класса соответствует идентификатору языка Плэнер, а унарные операции «-», «~» и «*», примененные к нему, возвращают объекты классов, соответствующие обращениям к переменным с префиксами «:», «.» и «*». За счет использования оператора «!» для последующего сегментирования обращения к переменной выражения языка C++ получаются внешне достаточно похожими на конструкции Плэнера.

Таким образом, если создать объекты L и P классов `PlnLListConstructor` и `PlnPListConstructor` соответственно и объекты класса `PlnSymbol` для каждого используемого в программе идентификатора, то можно будет провести следующие аналогии между выражениями на языке C++ и выражениями на языке Плэнер:

Выражение C++	Выражение Плэнера
`(L| 2, 3, 4)`	`(2 3 4)`
`(L| 2, ~X, !~Y)`	`(2 .X !.Y)`
`(P| EQ, 3, 4)`	`[EQ 3 4]`
`(P| IS, (L| 2, *Y, 4), ~X)`	`[IS (2 *Y 4) .X]`
`!(P| 2, 3, 4)`	`<2 3 4>`

3.2 Вычисление выражений и режим возвратов

В языке Плэнер существует возможность режима возвратов (бэктрекинга): в программе можно поставить *развилку*, организующую перебор. Если в дальнейшем вычисление заходит в тупик, то вырабатывается *неуспех*, программа полностью восстанавливает свое состояние на момент развилки, выбирает другой вариант вычислений и продолжает свою работу. То есть для реализации режима возвратов необходима возможность в любой момент восстановить ранее сохраненное состояние программы и продолжить вычисление с восстановленной точки. Поэтому, в отличие от Лиспа, вычисление выражений Плэнера нельзя реализовывать рекурсивно.

Для итеративного вычисления выражений было решено промоделировать вычисление с помощью модифицированного стека функций, существовавшее в реализации Плэнера для системы БЭСМ-6 [8]. В этой реализации фреймы функций, устанавливающих точку развилки, помечались особым способом, и ни они, ни фреймы, расположенные выше по стеку, не удалялись из стека после возврата из них. При установке развилки сохранялось полное состояние программы на тот момент, не исчезающее даже после завершения функции, установившей развилку. При дальшейшем возникновении в программе неуспеха происходил переход на существующий фрейм, хранящий правильный лексический контекст и точку возврата. Отдельно хранились обратные операторы, которые должны были быть выполнены при возврате для восстановления корректного состояния программы.

Для моделирования такого вычисления в рамках библиотеки классов языка C++ был создан класс `PlnEvaluator`, полностью отвечающий за вычисление выражений. Он хранит в себе указатель на двусвязный список фреймов функций, указатели на текущий и последний фреймы этого списка, список развилок, связанных с данным стеком и результат последнего вычисления (успешный либо неуспешный).

Каждый фрейм содержит в себе вычисляемое выражение, лексический контекст, массив для хранения вычисленных аргументов функций и адрес возврата, который может не совпадать с адресом предыдущего фрейма, поскольку некоторые уже завершившиеся фреймы не удаляются из стека.

При реализации перебора некоторые участки программы могут вычисляться несколько раз, то есть каждая функция должна уметь продолжать свое вычисление с конкретной точки, независимо от того, сколько аргументов уже было вычислено. Для этого фрейм функции хранит число — номер параметра, которым это выражение является. При вычислении аргументов функции или элементов списка для каждого i-го элемента создается новый стековый фрейм, и ему на хранение передается число i. Когда фрейм возвращает управление выражению, создавшему его, он, кроме результата своего вычисления, возвращает и это число. Изменяя это число при возврате к фрейму по несупеху, можно заставить функцию повторно вычислить какой-то ее аргумент.

Работа с развилками осуществляется через класс `PlnEvaluator`, позволяющий добавить развилку, соответствующую текущему вычисляемому фрейму, или удалить последнюю развилку. В последнем случае активной развилкой становится предыдущая, и при возникновении неуспеха управление будет передано ее фрейму.

При откате по неуспеху программа должна полностью восстановить свое состояние, сохраненное при установке последней развилки. При этом восстановление вычисляемого выражения, его адреса возврата и номера параметра, который оно должно будет вернуть родительскому фрейму, осуществляется за счет того, что нужный фрейм не удаляется из стека. Но значения локальных и глобальных переменных могли быть изменены на неуспешной ветви вычисления. Поэтому все действия программы на неуспешном пути запоминаются, и при откате выполняются противоположные действия. В каждом объекте класса `PlnStackFork` хранится список *обратных операторов* — действий, которые должны быть выполнены при откате к этой развилке. При возврате к развилке необходимо перебрать элементы списка и последовательно восстановить старые значения переменных, хранящиеся в них.

При такой реализации для вычисления выражения языка Плэнер, записанного в виде конструкций языка C++, в программе должен быть создан объект класса `PlnEvaluator`, соответствующий стеку фреймов функций. При вызове метода `Evaluate` этого класса с нужным выражением, представленным объектом класса `PlnReference`, в качестве параметра, в списке создается новый фрейм, соответствующий этому выражению, и начинается циклическое вычисление фреймов стека. Поскольку в процессе вычисления возможно создание новых и удаление отработавших фреймов, то цикл будет работать, пока из стека не будет удален фрейм исходного выражения, выдав конечный результат. При этом метод `Evaluate` никогда не будет вызван рекурсивно.

При вычислении выражения, находящегося в некотором стековом фрейме, используются результат последнего вычисления, количество вычисленных параметров (при нормальном ходе вычисления без развилок оно совпадает с количеством попыток вычисления данного фрейма), значения вычисленных параметров. Само вычисление осуществляется следующим образом:

- L-список: если количество вычисленных параметров меньше, чем длина списка, то в стек добавляется фрейм для вычисления очередного элемента списка, иначе из вычисленных значений формируется результирующий список с учетом возможной сегментированности элементов.

- P-список: первым элементом в нем может быть либо имя функции, являющееся идентификатором, либо обращение к переменной с префиксом «.» или «:». В последнем случае сразу же берется значение переменной из ее идентификатора или лексического контекста фрейма. Полученный таким образом идентификатор содержит значение — объект класса, представляющего функциональные объекты. Далее вычисление зависит от конкретного функционального объекта. При вычислении функций создаются новые стековые фреймы для параметров, которые должны быть вычислены, создается новый лексический контекст, в котором осуществляется связывание формальных параметров с фактическими, и либо добавляется новый фрейм для тела функции, либо вызывается метод, содержащий тело функции в виде кода на C++.

- «.»-переменная: возвращается S-выражение, связанное с соответствующим ей идентификатором в лексическом контексте, хранящемся в фрейме вычисления.
- «:»-переменная: возвращается S-выражение, хранящееся внутри идентификатора, соответствующего переменной.
- Сегментированные выражения не могут быть вычислены, попытка их вычисления вызывает ошибку.
- Для всех остальных типов S-выражений значениями являются они сами.

Если вычисление значения выражения, соответствующего текущему стековому фрейму, было успешно завершено, то возвращается результат вычисления и число — номер вычисленного параметра, которые затем могут быть использованы следующим вычисляемым фреймом. Затем выполняется проверка, можно ли удалять вычисленный фрейм из стека, и если ниже него не было фреймов с развилками, то он удаляется. Текущим становится фрейм, находившийся в адресе возврата вычислившегося.

Если же в результате вычисления выражения возник неуспех, то никакой результат не возвращается и происходит откат к последней развилке: ее фрейм становится текущим, а все, находящиеся в стеке после него, удаляются. При откате последовательно выполняются все сохраненные обратные операторы, которые затем удаляются.

3.3 Сопоставление выражения с образцом

В языке Плэнер сопоставление бывает двух типов: выражения с образцом и образца с образцом. Первое сопоставление обычно осуществляется при вызове функции `IS` или при поиске утверждения в базе данных. Второе возникает только при вызове теорем по образцу.

При сопоставлении выражения с образцом бывает необходимо вызывать функции и *сопоставители* — процедуры, осуществляющие расширенное сопоставление. При вычислении функций могут возникать неуспехи, которые должны быть отслежены, и создаваться обратные операторы, причем эти операторы должны быть сохранены даже при удачном сопоставлении. Поэтому сопоставление с образцом необходимо реализовывать тоже итеративно, с использованием стековых фреймов.

При реализации были введены новые типы стековых фреймов, отвечающие за сопоставление выражения с образцом. При сопоставлении выражения-списка с образцом, содержащим сегментированные выражения, возникает необходимость перебора элементов списка, чтобы найти сегмент, соотвествующий элементу образца. Поэтому для оптимизации сопоставления было решено использовать два типа фреймов: один, вычисление которого достаточно просто и быстро, для сопоставления атомарных выражений, и второй, в котором уже может возникать перебор, для L-списков. Сам перебор организован на основе режима возвратов: если выбранный в какой-то момент сегмент списка приводит к неудачному сопоставлению, в программе возникает неуспех, осуществляется возврат на фрейм, выбравший этот сегмент, и сопоставление продолжается с увеличенным сегментом списка.

Такое решение не вполне корректно, так как с точки зрения языка режим возвратов и механизм сопоставления являются совершенно независимыми, но оно позволяет упростить сопоставление. Перебор элементов списка должен осуществляться в рамках циклического вычисления стековых фреймов, поэтому, если не использовать имеющуюся реализацию режима возвратов, потребовалось бы создать полностью аналогичный ему механизм перебора, то есть задублировать функциональность бектрекинга, либо использовать механизм исключений языка C++, что сильно ухудшило бы производительность. Кроме того, гарантируется, что неуспех, возникший при неудачном переборе, будет обработан фреймом-сопоставителем или функцией, начавшей осуществлять сопоставление, то есть механизмы возвратов и сопоставления оказываются связанными только на уровне реализации, но не на уровне семантики программы на языке Плэнер.

Во фреймах, осуществляющих сопоставление атомарных выражений, может возникнуть обращение к функции-сопоставителю. Такое обращение обрабатывается как вычисление обычной функции. Разница заключается только в том, что для тела функции создается фрейм сопоставления, а не вычисления, и к списку фактических параметров функции, кроме значений,

соответствующих формальным параметрам, добавляется выражение, свойства которого должна проверить функция-сопоставитель. То есть функциям-сопоставителям всегда передается не меньше одного аргумента, что особенно важно при реализации встроенных сопоставителей на языке C++.

3.4 Встроенная база данных

Встроенная база данных языка Плэнер является по сути просто набором утверждений (L-списков, состоящих из атомов) с возможностью выбора тех из них, которые соответствуют конкретному образцу. Поэтому при имеющемся сопоставлении выражения с образцом реализация базы данных требует только написания трех основных встроенных функций для работы с ней: добавляющей новое выражение в массив утверждений, удаляющей выражение из массива и функции поиска. Функция поиска по базе данных должна перебирать все имеющиеся утверждения и последовательно сопоставлять их с заданным образцом (добавлять в стек фрейм для такого сопоставления). При успешном сопоставлении она должна завершаться, при неуспешном — откатывать все изменения, которые могли быть сделаны при предыдущем сопоставлении, и выбирать следующее утверждение.

Для оптимизации перебора утвержений из базы данных, которых может быть достаточно много, массив с ними хранится в классе-контейнере `DataBase`, имеющем метод `SearchSimilar(pattern)` для выбора утверждений, которые могут соответствовать образцу. Этот класс не имеет доступа к стеку вычисления функций и поэтому не может осуществить полное сопоставление выражений с образцом, но он производит частичное сопоставление: проверяет соответствие длин образца и утверждения; если в образце заданы атомы, то проверяется наличие этих атомов в утверждении. После проверки таких мягких условий список утверждений сокращается, и впоследствии требуется осуществлять полное сопоставление меньшего числа выражений.

3.5 Вызов процедур по образцу

В языке Плэнер существует еще один вид процедур — *теоремы*. Они отличаются от обычных функций тем, что вызываются не по имени, а по образцу, который играет роль и имени, и набора формальных параметров. В программе может существовать несколько процедур с одинаковыми или похожими образцами, что позволяет перебирать их по очереди, применяя ту, которая сумеет найти решение.

Любая теорема вычисляется так же, как и встроенная функция `PROG`, то есть у нее есть список локальных переменных и тело — набор последовательно вычисляющихся выражений. Поэтому вычисление уже выбранной теоремы реализовывается довольно просто на основе уже введенных конструкций.

Основная сложность при реализации теорем заключается в организации сопоставления образцов между собой. И вызывающий образец, и образец теоремы являются полноценными образцами, поскольку, например, и в одном, и в другом могут содержаться обращения, меняющие значения переменных: в одном для получения результата работы теоремы, в другом — для передачи параметров.

Сопоставление образцов между собой является в общем случае алгоритмически неразрешимой задачей [9]. Как минимум, если в каждом образце встречаются обращения к процедурам-сопоставителям, то их корректное сопоставление аналогично проверке эквивалентности алгоритмов. Если запретить использование сопоставителей, то все равно возникают проблемы из-за наличия сегментных образцов. Поэтому обычно для реализации такого сопоставления на образцы накладывают ограничения.

В описании языка Planner [2] эта проблема реализации никак не регламентируется. В диалекте Плэнер в образцах, сопоставляемых друг с другом, запрещено использовать обращения к любым процедурам (и функциям, и сопоставителям) и сегментные образцы, при этом вложенность списков и простые обращения к переменным не ограничиваются. Этого достаточно и для решения большинства задач, и для эффективной реализации.

Другой проблемой, возникающей при сопоставлении образцов, является осуществление связывания переменных. Если сопоставляются две переменные, ни одна из которых не имеет зна-

чения, между ними должна возникнуть связь: если в дальнейшем одна из этих переменных получит значение, то это значение должно быть присвоено и другой переменной. Более того, если переменная без значения сопоставляется со списком, среди элементов которого есть другая переменная, то на значение первой переменной накладываются ограничения. Например, если производится сопоставление `[MATCH *X (A *Y B)]`, то на переменную X будет наложено ограничение, что ее значение является списком, первый и третий элементы которого определены, а второй должен будет совпадать со значением переменной Y.

Эти ограничения можно рассматривать как каркас значения переменной, «полуфабрикат» [8]. Если ввести новые структуры данных для хранения этого полусформированного значения, то можно сделать его недоступным пользователю системы, но использовать в сопоставлениях наравне с другими выражениями. При этом проверка непротиворечивости ограничений на значение переменной сводится к сопоставлению имеющегося у переменной значения-«полуфабриката» с новым ограничением.

Для реализации таких «полуфабрикатов» вводится класс `PlnHalfFormed`, содержащий внутри себя «умный» указатель на какое-либо выражение. Если в каком-либо объекте этого класса этот указатель пустой, то объект считается неопределенным и может впоследствии получить любое значение. Если же указатель указывает на выражение-список, то у этого списка тоже могут быть не окончательно сформированные элементы. При сопоставлении переменных производится сопоставление таких полусформированных значений. При сопоставлении переменных, не имеющих ни настоящего, ни полусформированного значения, создается новый объект класса `PlnHalfFormed`, который становится значением обеих переменных. Таким образом осуществляется связь между ними. Если же программе необходимо получить настоящее значение переменной, например, являющейся аргументом функции, то проверяется, указывает ли указатель из объекта класса `PlnHalfFormed` на полностью сформированное выражение, и если да, то значение, связанной с переменной обновляется.

Следует отметить, что полусформированные значения могут быть только у локальных переменных, так как использование глобальных переменных в сопоставлениях полностью равносильно использованию их значений. Глобальные переменные не могут получить или изменить свое значение ни при каком сопоставлении.

4 Пример использования библиотеки

Полученная библиотека классов позволяет записать программу на языке Плэнер в виде набора выражений языка C++ и вычислить ее. Рассмотрим пример программы на Плэнере, которая вводит новый сопоставитель, проверяющий, является ли список палиндромом:

```
[DEFINE PALYNDROM (KAPPA ()
    [AUT ()                                ; либо пустой список
         [LIST 1]                          ; либо список из одного элемента
         [SAME (X) (*X <PALYNDROM> .X)]    ; либо палиндром
    ]
)]

[IS [PALYNDROM] (A 5 A)]                   ; -> T
```

Аналогичная программа на языке C++ с использованием библиотеки:

```
// объект, осуществляющий вычисления
PlnEvaluator stack;

// объекты, конструирующие списочные выражения
PlnLListConstructor L;
PlnPListConstructor P;

// используемые в программе идентификаторы
PlnSymbol PALYNDROM("PALYNDROM"), X("X"), A("A");
```

```
stack.Evaluate(
(P| DEFINE, PALYNDROM, (L| KAPPA, ~L,
     (P| AUT, ~L,
              (P| LIST, 1),
              (P| SAME, (L| X), (L| *X, !(P| PALYNDROM), ~X))
     )
)));

PlnReference result = stack.Evaluate(
    (P| IS, (P| PALYNDROM), (L| A, 5, A)));
```

После работы этой программы переменная `result` будет содержать в себе S-выражение, соответствующее символу `T`.

Объекты класса `PlnSymbol` должны создаваться для всех идентификаторов, имеющихся в программе, кроме описанных в библиотеке. К последним относятся атомы `T`, `LAMBDA`, `KAPPA`, `ANTEC`, `CONSEQ` и `ERASING`. Атом `T` появляется в библиотеке как результат работы функций, возвращающих логические значения. Остальные пять атомов используются функцией `DEFINE` для определения типа создаваемой процедуры.

Символы `DEFINE`, `IS`, `AUT`, `SAME`, `LIST` являются именами встроенных функций и сопоставителей и тоже вводятся в библиотеке.

5 Вспомогательное программное обеспечение

Созданные средства конструирования и вычисления выражений языка Плэнер позволяют реализовать интерактивный интерпретатор языка. Ядром интерпретатора является класс `IntelibPlnLoop`, осуществляющий основной цикл работы: считывание выражения из файла или потока стандартного ввода, вычисление полученного выражения с помощью объекта класса `PlnEvaluator` и вывод текстового представления результата на экран.

Считывание выражения и его лексический и синтаксический анализ осуществляются с помощью средств, предоставляемых библиотекой InteLib. В библиотеке имеется класс `IntelibGenericReader`, отвечающий за синтаксический анализ языков, основанных на S-выражениях. Он может быть перепрограммирован для конкретного языка с учетом его особенностей. В частности, для реализации чтения выражений языка Плэнер от этого класса наследуется класс `PlannerReader`, в конструкторе которого указывается, что символом комментария является «;»; списки могут быть ограничены скобками вида (), [] и <> (для каждого типа скобок указывается конкретная функция, умеющая формировать из отдельных элементов списка требуемые структуры данных); префиксы перед идентификаторами должны сформировать из них обращения к соответствующим переменным.

Благодаря перегруженным операциям и введенным классам для конструирования списков языка Плэнер, выражения на языке C++ являются синтаксически похожими на выражения Плэнера. Тем не менее, синтаксис двух языков отличается, и недостаточно просто написать подпрограмму на языке Плэнер для включения ее в большой проект. Ее надо переписать с учетом синтаксиса языка C++, что может быть неудобно из-за большого количества вспомогательных символов.

Поэтому удобным средством является транслятор программы на языке Плэнер в модуль на языке C++. При трансляции выражения Плэнера переводятся в аналогичные им конструкции, вдобавок в модуль добавляются все необходимые объявления переменных. Создаются объекты класса `PlnEvaluator` для осуществления вычисления, классов `PlnPListConstructor` и `PlnLListConstructor` для конструирования списков, класса `PlnSymbol` для всех используемых в программе идентификаторов.

Модуль на языке C++, содержащий программу языка Плэнер, генерируется автоматически этим транслятором, но тем не менее может быть прочитан и понят человеком, поскольку выражения конструируются все теми же удобными синтаксическими способами. Он может быть

подключен к основному проекту на языке C++, а остальные модули этого проекта требуют написания минимального количества кода с использованием библиотеки. В большинстве ситуаций будет достаточно написать несколько P-списков с вызовами функций, реализованных в программе Плэнера, и получить результат их работы в виде S-выражений.

Транслятор сам написан на языке Плэнер. Для его запуска можно использовать описанный выше интерпретатор. При этом из текста самого транслятора тоже можно создать модуль на языке C++. Таким образом, запуская транслятор из-под интерпретатора и применяя его к собственному коду, можно получить модуль на языке C++.

Основная программа, написанная на C++, анализирует переданные ей через командную строку параметры, среди которых, кроме имени транслируемого файла, могут быть управляющие директивы, инициализирует модуль транслятора и вызывает его основную функцию, передавая ей в качестве параметра имя файла.

Результатом компиляции основной программы вместе со сгенерированным модулем является исполняемый файл, являющийся транслятором из языка Плэнер в код на языке C++. Этот файл с одной стороны использует код на языке Плэнер для своей работы, но с другой — работает намного быстрее, чем транслятор, запущенный из-под интерпретатора.

Работа самого транслятора состоит из следующих шагов:

1. Считывание выражений программы на языке Плэнер, одновременно с этим происходит обработка директив.

2. Первый проход по программе: имена идентификаторов преобразовываются в соответствии с правилами замены символов, все используемые в программе идентификаторы заносятся в общую таблицу.

3. Генерация заголовочного файла. В заголовочном файле создается класс, инкапсулирующий объявления переменных для идентификаторов программы. Для каждого выражения в классе объявляется отдельный метод, в котором и будет происходить его вычисление. Конструктор этого класса должен поочередно вызвать эти методы, выполнив таким образом всю программу. Объявляется функция инициализации модуля

4. Второй проход: генерация файла реализации. Выражения языка Плэнер преобразуются в соответствующие им конструкции языка C++. Каждое выражение вычисляется объектом класса `PlnEvaluator` в соответствующем ему методе. Генерируется код для конструктора класса, вызывающего методы для всех выражений, и функции инициализации модуля. При вызове этой функции из основной программы будет создан объект класса, описанного выше, вызван его конструктор и вычислится подключаемая программа Плэнера.

При первом проходе по тексту программы транслятором осуществляется замена имен идентификаторов, если это необходимо. Алфавит языка Плэнер намного шире алфавита языка C++. При этом для любого идентификатора, используемого в программе, должна существовать переменная класса `PlnSymbol`, имя которой должно соответствовать правилам записи идентификатором языка C++. Так что допустимые в программе на Плэнере имена `=A+B=`, `*`, `A+-`, `/Zzz` должны быть преобразованы к другому виду, причем однозначным образом.

Для такого преобразования был использован способ, применяющийся в библиотеке InteLib для идентификаторов языков Lisp и Scheme. Он основан на том, что в этих языках не различаются заглавные и строчные символы, в отличие от языка C++. Поэтому при записи переменных для латинских букв из имен идентификаторов используются только заглавные латинские буквы, а для специальных символов — их названия, записыванные строчными буквами.

Таким образом, каждому идентификатору из программы соответствует корректный уникальный идентификатор языка C++. Например:

Идентификатор языка Плэнер	Переменная языка C++
`TheAtomName`	`THEATOMNAME`
`=A+B=`	`equalAplusBequal`
`*`	`star`
`A+-`	`Aplusminus`
`/Zzz`	`backslZZZ`

Для управления работой транслятора в программе на Плэнере могут быть использованы некоторые директивы. Они позволяют задавать правила перевода специальных символов в именах идентификаторов; идентификаторы, переменные для которых уже существуют в библиотеке; имена, используемые для генерируемых заголовочных и исходных файлов; имя функции инициализации модуля, которая должна будет быть вызвана из основной программы на C++; классы для встроенных функций и некоторые другие параметры работы транслятора.

Список литературы

[1] Hewitt C. *PLANNER: A Language for Proving Theorems in Robots.* IJCAI 1969.

[2] Hewitt C. *Description and theoretical analysis (using schemata) of PLANNER: a language for proving theoremes and manipulating models in a robot.* TR-258, AI Lab., MIT, 1973.

[3] E. Bolshakova and A. Stolyarov. *Building functional techniques into an object-oriented system.* In Knowledge-Based Software Engineering. Proceedings of the 4th JCKBSE, vol. 62 of Frontiers in Artificial Intelligence and Applications, pages 101-106, Brno, Czech Republic, September 2000. IOS Press, Amsterdam, 2000.

[4] И. Г. Головин, А. В. Столяров. *Объектно-ориентированный подход к мультипарадигмальному программированию.* Вестник МГУ, сер. 15 (ВМиК), N 1, 2002 г., стр. 46–50.

[5] Andrey V. Stolyarov. *A framework of heterogenous dynamic data structures for object-oriented environment: the S-expression model* In Knowledge-Based Software Engineering. Proceedings of the 6th JCKBSE, vol.108 of Frontiers in Artificial Intelligence and Applications, pages 75–82, Protvino, Russia, August 2004. IOS Press.

[6] Официальный сайт библиотеки InteLib http://www.intelib.org

[7] В.Н.Пильщиков. *Язык программирования ПЛЭНЕР-БЭСМ.* Издательство Московского университета, 1978.

[8] В.Н.Пильщиков. *Система программирования ПЛЭНЕР-БЭСМ.* Издательство Московского университета, 1983.

[9] В.Н.Пильщиков. *Язык Плэнер.* М.:Наука, 1983.

[10] J. McCarthy. *Recursive functions of symbolic expressions and their computation by machine.* Communications of the ACM, 3(4): p. 184–195, Apr 1960.

УДК 519.816

ОБОСНОВАНИЕ МЕТОДА РАЗУМНЫХ ЦЕЛЕЙ ДЛЯ ЗАДАЧ С НЕОПРЕДЕЛЕННОСТЬЮ

© 2008 г. А. В. Холмов

axely@yandex.ru

Кафедра Системного анализа[1]

1 Введение

Рассмотрим задачу выбора решения, имеющую большое число практических приложений. Пусть есть конечное число N альтернативных вариантов решения (при этом оно может быть сколь угодно большим). Каждый вариант задан значениями m характеристик (то есть показателями качества) $y_1, y_2, \ldots, y_m$. Необходимо найти наиболее предпочтительный вариант среди предложенных. Значения показателей для альтернатив могут быть, например, результатами эксперимента (быть может, вычислительного) либо результатами наблюдений и сбора статистики.

Будем исходить из того, что выбор наиболее предпочтительной альтернативы необходимо осуществить некоему лицу, принимающему решения (ЛПР). Основная задача, которая возникает при разработке систем поддержки принятия решений (СППР) — помочь ЛПР (либо другому пользователю СППР) в анализе возможностей принятия решения.

Необходимость применения СППР возникает в задачах выбора с большим числом вариантов. Как правило, человеческий разум не способен уже в задаче из 20 вариантов и 7 характеристик адекватно воспринимать информацию, представленную в виде списка альтернатив. Это приводит к затруднениям при ее практическом решении. Таким образом, требуется привести информацию в такую форму, которая была бы понятна и в то же время сохраняла бы все существенные черты исходной задачи.

Математически проблема выбора формулируется следующим образом. Пусть возможные альтернативные решения x являются элементами конечного множества X:

$$x \in X, |X| < \infty.$$

Пусть выбор осуществляется с использованием нескольких показателей качества (критериев выбора решений) $y = f(x)$:

$$y = f(x) = (f_1(x), f_2(x), \ldots, f_m(x)).$$

Функция f является отображением, переводящим исходное множество стратегий в множество точек критериального пространства. Ограничимся случаем $f : X \to \mathbb{R}^m$. Для определенности предположим, что предпочтительным является увеличение значений критериев.

Для поддержки принятия решения в этой задаче в 1994г. А.В. Лотовым и Д.В. Гусевым в [1] был предложен метод разумных целей (МРЦ), позволяющий выбрать подмножество точек множества X на основе представления совокупности критериальных точек $f(x), x \in X$, в графическом виде. МРЦ был успешно применен к задачам экологического и экономического характера.

МРЦ рассчитан на использование четко заданной информации: каждой альтернативе ставится в соответствие некоторая точка в критериальном пространстве. На практике же значения показателей качества для альтернатив часто заданы неточно, а значения того или иного критерия удается оценить отрезком, внутри которого находится это значение. В этом случае

[1]Работа была частично поддержана грантами Президента РФ по государственной поддержке ведущих научных школ (проект №НШ-2982.2008.1), РФФИ (проект №07-01-00472), программы фундаментальных исследований РАН №14 и программы фундаментальных исследований ОМН РАН №3.

каждой альтернативе соответствует не точка в m-мерном пространстве, а m-мерный параллелепипед:

$$A = \{y \in \mathbb{R}^m | \underline{y}_i \leqslant y_i \leqslant \overline{y}_i, i = \overline{1,m}\}.$$

МРЦ был перенесен на такие задачи в [2,3].

Задача данной работы состоит в том, чтобы дать обоснование МРЦ для случая неточной информации. При этом в разделе 2 на основе описания МРЦ для точных данных вводятся основные понятия, используемые в исследовании; в разделе 3 описывается модифицированный МРЦ для случая с неточной информации; в разделе 4 проводится анализ метода и приводится его обоснование; наконец, в разделе 5 рассматривается пример применения этого метода.

2 Метод разумных целей

Опишем сначала МРЦ для решения задач с данными, которые заданы точно, то есть когда нет неопределенности и каждой альтернативе ставится в соответствие некоторая точка в критериальном пространстве. Этот метод поддержки принятия решений основан на использовании компьютерной визуализации информации при отборе одного или нескольких предпочтительных вариантов решения из их большого числа для дальнейшего детального анализа и окончательного выбора решения.

Так как данные заданы точно, то можно считать, что конечное число альтернатив задано в виде строк некоторой таблицы, часть столбцов которой представляют собой значения критериев выбора. Особенностью МРЦ является интерактивная визуализация паретовой (недоминируемой) границы выпуклой оболочки многомерного множества точек, порождаемых строками таблицы.

Напомним, что для определенности рассматривается задача многокритериальной максимизации, то есть пользователь заинтересован в увеличении значений каждого из критериев.
Определение. Пусть $y = (y_1, y_2, \ldots, y_m)$ и $y' = (y'_1, y'_2, \ldots, y'_m)$ – некоторые векторы из $\mathbb{R}^m$: $y, y' \in \mathbb{R}^m$. Будем говорить, что y' доминирует y по Парето, если

$$y'_i \geqslant y_i, \forall i = \overline{1,m} \text{ и } y' \neq y.$$

Будем обозначать доминирование по Парето через $y' \succ y$.
Определение. Будем говорить, что y' доминирует y по Слейтеру, если

$$y'_i > y_i, \forall i = \overline{1,m}.$$

Будем обозначать доминирование по Слейтеру через $y' \succ_S y$.
Обозначим

$$Y = f(X) = \bigcup_{x \in X} f(x).$$

Пусть $Y = \{y^1, y^2, ..., y^N\}$.

В МРЦ изучается не само множество Y, а его выпуклая оболочка:

$$Y_C = conv(y^1, \ldots, y^N) = \{y \in \mathbb{R}^m | y = \sum_{i=1}^{N} \alpha_i y^i, \alpha_i \geqslant 0, \sum_{i=1}^{N} \alpha_i = 1\}.$$

Так как ЛПР заинтересован в увеличении оценок по всем критериям, то он сделает свой выбор не из всего исходного множества Y_C, а из его паретовского (недоминируемого по Парето) подмножества:

$$P(Y_C) = \{y \in Y_C | \nexists y' \in Y_C : y' \succ y\}.$$

Определение. Выпуклой оболочкой Эджворта-Парето (ВОЭП) множества Y называют множество

$$Y_C^P = \{y \in \mathbb{R}^m | \exists y' \in Y_C : y'_i \geqslant y_i, \forall i = \overline{1,m}\}.$$

Из определения видно, что $Y_C^P = Y_C + (-\mathbb{R}_+^m)$, где $\mathbb{R}_+^m = \{y = (y_1, ..., y_m) \in \mathbb{R}^m | y_i \geqslant 0, \forall i = \overline{1,m}\}$. Отметим, что $P(Y_C) = P(Y_C^P)$.

МРЦ состоит из следующих шагов:

1. построение (или аппроксимация) выпуклой оболочки Эджворта-Парето (ВОЭП) множества $Y = f(X)$;

2. визуализация паретовой границы ВОЭП наборами ее двукритериальных сечений;

3. выбор пользователем целевой точки на паретовой границе ВОЭП;

4. отбор альтернатив, показатели качества которых соответствуют в определенном смысле целевой точке.

Методы аппроксимации выпуклых тел, применяемые на первом шаге, описаны в [4].

Рассмотрим проблему визуализации. При $m = 2$ недоминируемая граница Y_C показывает связь между критериями в графической форме и поэтому характеризует замещение (так называемая кривая объективного замещения). Проанализировав кривую объективного замещения, пользователь может выбрать наиболее предпочтительное сочетание значений критериев (цель) прямо на недоминируемой границе ВОЭП. Ясно, что цель, вообще говоря, не совпадает ни с одним из критериальных векторов из Y, так что требуется дополнительна процедура отбора.

Если число критериев равно трем, то для демонстрации недоминируемой границы ВОЭП можно использовать более сложные изображения, сохраняющие в то же время простоту. Это так называемые карты решений, представляющие собой совокупности границ сечений ВОЭП, определяемых значениями третьего критерия. Эти границы не пересекаются, что определяет их близость к обычным топографическим картам.

Если же число критериев превосходит три, то необходимо использовать программное обеспечение диалоговых карт решений (ДКР), в которых влияние четвертого, пятого и др. критериев можно изучить с помощью интерактивной визуализации и анимации карт решений. Выбрав одну из карт решений, пользователь может указать цель непосредственно на предпочтительном сечении. Подробно визуализация на основе ДКР описана в [4].

Так как обычно целевая точка y^* не совпадает ни с одним из критериальных векторов, то отбираются одна или несколько альтернатив, связанных с ней значениями критериев в том или ином смысле. В [1] было предложено отбирать альтернативы следующим образом: критериальная точка отобранной альтернативы должна быть максимумом по $y \in Y$ по крайней мере одной функции $U(y, \lambda)$, где $\lambda_i > 0, i = \overline{1, m}$ – параметры свертки критериев, построенной с использованием целевой точки y^*:

$$U(y, \lambda) = -\max_i \lambda_i (y_i^* - y_i).$$

Точнее говоря, для любой отобранной с помощью МРЦ альтернативы x' существует набор $\lambda' = (\lambda'_1, \lambda'_2, \ldots, \lambda'_m) > 0$ такой, что

$$f(x') \in Arg \max_{y \in Y} U(y, \lambda').$$

Реализовать такой подход "в лоб" не представляется возможным, так как пришлось бы решать бесконечное число задач оптимизации для всех $\lambda > 0, \sum_{i=1}^{m} \lambda_i = 1$. В [1] было показано, что совокупность решений всех таких задач оптимизации есть точки, выбранные по следующему алгоритму (см. рис. 1):

- Рассматривается множество $y^* + (-\mathbb{R}^m_+)$, которое содержит все точки, доминируемые целевой точкой y^*, и является конусом в $\mathbb{R}^m$ с вершиной в y^*.

- На конус $y^* + (-\mathbb{R}^m_+)$ проецируются точки множества Y, которые лежат вне его. Для этого у каждой точки y множества Y сравниваются значения координат со значениями соответствующих координат целевой точки y^*. Если значение координаты y_i лучше y_i^* (то есть $y_i > y_i^*$), то это значение заменяется на значение y_i^*; в противном случае значение координаты не меняется.

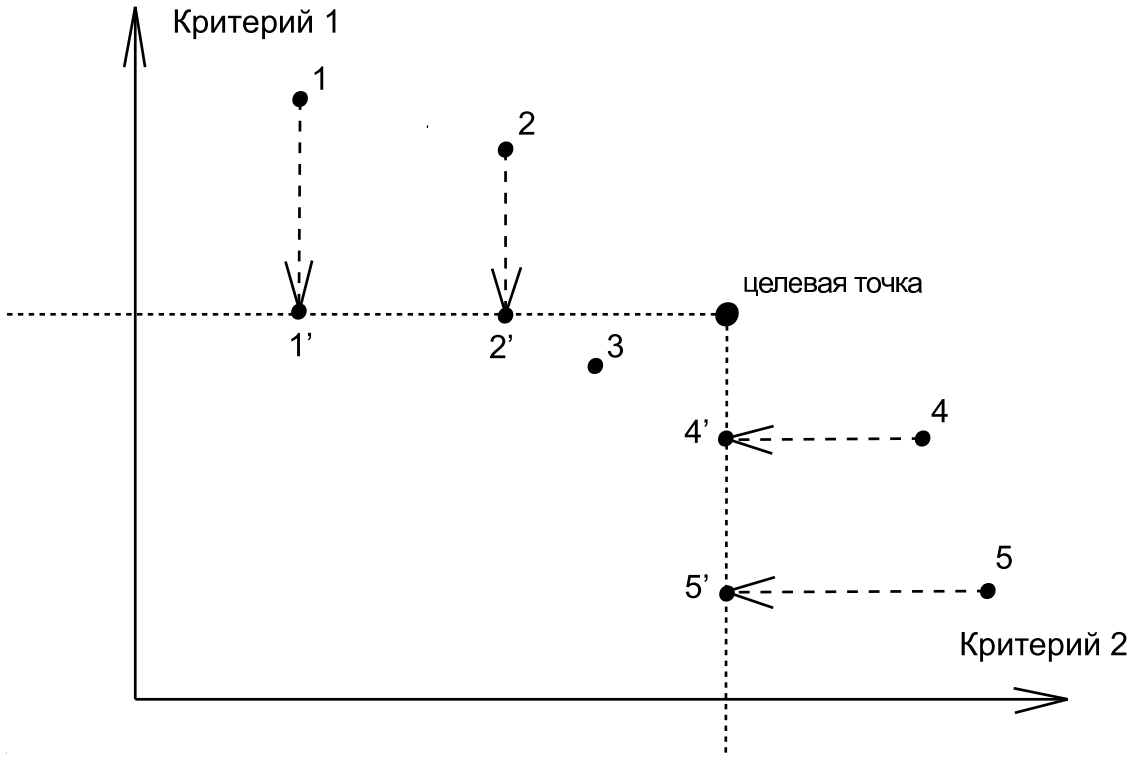


Рис. 1: Алгоритм отбора кртериальных точек по целевой точке для задачи без неопределенности

- Среди модифицированных критериальных точек выбираются недоминируемые по Парето. Отбираются альтернативы из X, которые порождают выбранные критериальные точки.

На рис.1 алгоритм отбора точек представлен для случая двух критериев. На нем видно, что точки $1, 2, 4, 5$ лежат вне конуса и проецируются на него. Точка 3 лежит внутри конуса, а потому не изменяется. Среди точек $1', 2', 3, 4', 5'$ недоминируемыми по Парето являются точки $2', 3, 4'$. Это значит, что альтернативами, отобранными на основе целевой точки, выбранной ЛПР, являются альтернативы, критериальными точками которых являются точки $2, 3, 4$.

Интерпретировать этот результат можно следующим образом: пользователь (ЛПР) фиксирует некоторый уровень притязания для каждого критерия, причем предполагается, что для него значительно важнее достичь этот уровень, чем превзойти его.

Заметим, что, вообще говоря, в совокупности отобранных критериальных точек могут быть точки, доминируемые по Парето (но не доминируемые по Слейтеру). При этом, однако, в такой совокупности будут содержаться и точки, доминирующие их по Парето. Так что для выделения недоминируемых по Парето точек необходимо провести дополнительный отбор среди полученных альтернатив.

3 Модификация МРЦ для случая неточной информации

Модификация МРЦ для случая неточной информации предложена в [2,3]. Каждой альтернативе x из множества X соответствует уже не точка в m-мерном пространстве, а m-мерный параллелепипед:

$$A = \{y \in \mathbb{R}^m | \underline{y}_i \leqslant y_i \leqslant \overline{y}_i, i = \overline{1,m}\}.$$

Через $Q(\mathbb{R}^m)$ обозначим множество всех параллелепипедов в пространстве $\mathbb{R}^m$, ограниченных гиперплоскостями, перпендикулярными осям координат. Тогда значения критериев задаются отображением $F : X \to Q(\mathbb{R}^m)$, то есть для $x \in X$ имеем $F(x) \in Q(\mathbb{R}^m)$.

Введем бинарные отношения для элементов множества $Q(\mathbb{R}^m)$, аналогичные доминированию по Парето и доминированию по Слейтеру для точек m-мерного пространства. Пусть A и B – некоторые m-мерные параллелепипеды: $A, B \in Q(\mathbb{R}^m)$.
Определение. Будем говорить, что A доминирует B по Парето, если

$$\text{для } \forall a \in A, \forall b \in B \text{ выполняется } a \succ b.$$

Будем обозначать доминирование по Парето для множеств также как и для точек через $A \succ B$. Аналогично дадим следующее
Определение. Будем говорить, что A доминирует B по Слейтеру, если

$$\text{для } \forall a \in A, \forall b \in B \text{ выполняется } a \succ_S b.$$

Будем обозначать доминирование по Слейтеру для множеств через $A \succ_S B$.
Модифицированный МРЦ для задачи с неточной информацией состоит из следующих шагов:

1. Построение (или аппроксимация) ВОЭП множества наилучших точек параллелепипедов, соответствующих всем альтернативам (то есть точкек $\overline{y} = (\overline{y_1}, ..., \overline{y_m})$ с наибольшими значениями координат);

2. Визуализация паретовой границы ВОЭП на основе использования ДКР;

3. Выбор пользователем целевой точки на паретовой границе ВОЭП;

4. Отбор альтернатив, показатели качества которых соответствуют в некотором определенном смысле целевой точке y^*.

Подчеркнем, что пункты 1-3 полностью повторяют эти пункты для случая без неопределенности. Алгоритм отбора альтернатив по выбранной ЛПР целевой точке состоит в следующем:

- На множество $y^* + (-\mathbb{R}^m_+)$, где y^* – целевая точка, проецируем m-мерные параллелепипеды, которые лежат вне его. Стоит заметить, что если часть параллелепипеда лежит вне построенного конуса, а часть внутри, то проецируются только те точки, которые находятся вне конуса. Точки, которые находятся внутри конуса остаются без изменений. Проецирование точек происходит по такому же алгоритму, что в детерминированном случае.

- Среди модифицированных множеств выбираются недоминируемые по Парето. Альтернативы, которые им соответствуют, и есть множество альтернатив, отбираемых для дальнейшего анализа.

Для случая двух критериев алгоритм проиллюстрирован на рис.2.
Как уже упоминалось, в [1] было доказано, что все решения задачи с точно заданными данными, полученные с помощью МРЦ, являются решениями задачи оптимизации для какой-либо из сверток критериев. Приведем анализ модифицированного МРЦ, в ходе которого будет доказано аналогичное утверждение, а, следовательно, дано обоснование отбора альтернатив в предложенном методе.

4 Анализ модифицированного МРЦ

Введем некоторые обозначения. Пусть

$$F(X) = \bigcup_{x \in X} F(x), Y = \{y \in \mathbb{R}^m | \exists x \in X : y \in F(x)\}.$$

Обратим внимание на то, что $Y \subset \mathbb{R}^m, F(X) \subset Q(\mathbb{R}^m)$.
Рассмотрим некоторую числовую функцию $U(\cdot)$, заданную на $\mathbb{R}^m$, то есть $U(\cdot) : \mathbb{R}^m \to \mathbb{R}^1$.

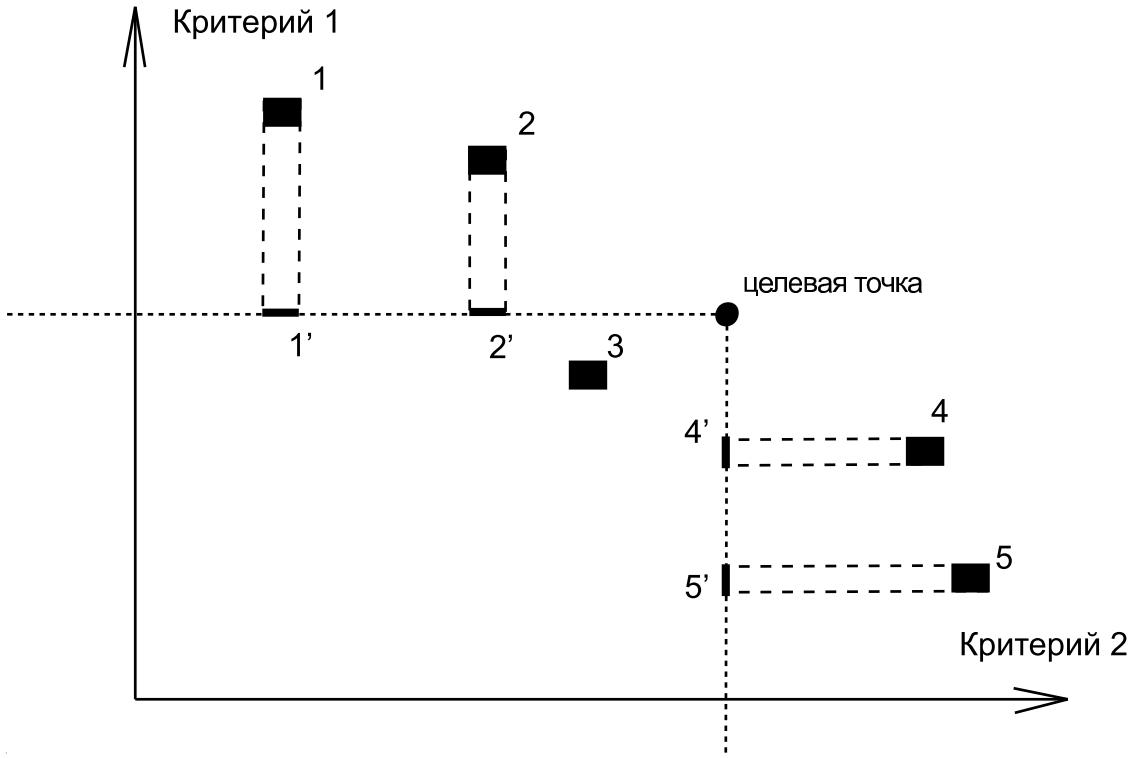


Рис. 2: Алгоритм отбора альтернатив по целевой точке для задачи с неточными данными

Определение. $U(\cdot)$ назовем строго монотонной функцией, если
для $\forall a, b \in \mathbb{R}^m : a \succ_S b$ имеем $U(a) > U(b)$,
для $\forall a, b \in \mathbb{R}^m : a \succ b$ имеем $U(a) \geqslant U(b)$.

Для удобства в дальнейшем будем использовать следующие обозначения:

$$U(A) = \bigcup_{y \in A} U(y), \max U(A) = \max_{y \in A} U(y), \min U(A) = \min_{y \in A} U(y).$$

Пусть $\mathcal{K}_M^{Y,y^*}$ – класс строго монотонных функций, у которых максимум на множестве $Y \cup \{y^*\}$ достигается в целевой точке $y^* \in \mathbb{R}^m$. Рассмотрим непустой подкласс функций $\mathcal{K}$ класса $\mathcal{K}_M^{Y,y^*}$.

Определение. Обобщенно близкими параллелепипедами из $F(X)$ к цели y^* относительно класса функций $\mathcal{K}$ назовем все $A' \in F(X) : \exists U'(\cdot) \in \mathcal{K}$, где

$$\max U'(A') \geqslant \min U'(A), \forall A \in F(X).$$

Будем обозначать совокупность обобщенно близких вариантов из $F(X)$ к цели y^* относительно класса функций $\mathcal{K}$ через $C(F(X), \mathcal{K}, y^*)$.

Рассмотрим следующую задачу. Пусть
$G \subset Q(\mathbb{R}^m)$ — некоторое подмножество множества m-мерных параллелепипедов;
$U'(\cdot) : \mathbb{R}^m \to \mathbb{R}$ — некоторая числовая функция.

Требуется найти все $A' \in G$ такие, что

$$\max U'(A') \geqslant \min U'(A), \text{для } \forall A \in G. \tag{1}$$

Тогда обобщенно близкие варианты из $F(X)$ к цели y^* относительно класса функций $\mathcal{K}$ — это объединение всех решений задачи (1) по всем функциям $U(\cdot) \in \mathcal{K}$ на множестве $F(X)$.

Так как множество $F(X)$ конечно, то решение каждой из задач (1) на этом множестве можно найти, например, простым последовательным сравнением наибольших и наименьших значений функции на всех элементах $F(X)$. Сложность заключается в том, что число функций $U(\cdot)$ в классе $\mathcal{K}$ может быть бесконечным.

Определение. Бинарное отношение $\mathcal{R}$ назовем соответствующим тройке $(F(X), \mathcal{K}, y^*)$, где $\mathcal{K} \subseteq \mathcal{K}_M^{Y,y^*}$, если

$$A\mathcal{R}B \Leftrightarrow \min U(A) > \max U(B), \text{ для } \forall U(\cdot) \in \mathcal{K}.$$

Свойства бинарного отношения $\mathcal{R}$:

1. Иррефлексивность

   Доказательство. Ясно, что для $\forall A \in F(X)$ и для $\forall U(\cdot) : \mathbb{R}^m \to \mathbb{R}$ выполняется, что $\min U(A) \leqslant \max U(A)$. Отсюда по определению бинарного отношения $\mathcal{R}$ следует, что $A\overline{\mathcal{R}}A$. Следовательно, $\mathcal{R}$ – иррефлексивное бинарное отношение.

2. Транзитивность

   Доказательство. Пусть $A\mathcal{R}B, B\mathcal{R}C$. Тогда для $\forall U(\cdot) \in \mathcal{K}$

   $$\begin{cases} \min U(A) > \max U(B), \\ \min U(B) > \max U(C). \end{cases}$$

   Отсюда следует, что

   $$\min U(A) > \max U(B) \geqslant \min U(B) > \max U(C).$$

   В свою очередь, отсюда следует, что

   $$\min U(A) > \max U(C).$$

   Отсюда по определению $A\mathcal{R}C$. Следовательно, $\mathcal{R}$ – транзитивное бинарное отношение.

Замечание. Свойства 1 и 2 бинарного отношения $\mathcal{R}$ показывают, что $\mathcal{R}$ является бинарным отношением строгого порядка.

Определение. Элемент $A \in F(X)$ называется $\mathcal{R}$-максимальным на $F(X)$, если $B\overline{\mathcal{R}}A, \forall B \in F(X)$.

Совокупность $\mathcal{R}$-максимальных элементов на $F(X)$ будем обозначать через $\mathbf{K}_{\mathcal{R}}(F(X))$.

Определение. Говорят, что отношение $\mathcal{R}$ обладает свойством фон Неймана-Моргенштерна (НМ-свойством) на $F^*(X) \subset F(X)$, если $\forall A \in F(X)$ либо $A \in F^*(X)$, либо найдется $B \in F^*(X)$ такой, что $B\mathcal{R}A$.

Определение. Если отношение $\mathcal{R}$ обладает свойством НМ-свойством на $\mathbf{K}_{\mathcal{R}}(F(X))$, то $\mathbf{K}_{\mathcal{R}}(F(X))$ называется решением по фон Нейману-Моргенштерну (ядром по фон Нейману-Моргенштерну) для бинарного отношения $\mathcal{R}$ на $F(X)$.

Справедливо следующее утверждение [5].

Теорема. Пусть $F(X)$ конечно. Тогда $\mathbf{K}_{\mathcal{R}}(F(X))$ не пусто и является решением по фон Нейману-Моргенштерну для отношения $\mathcal{R}$ на $\mathbf{K}_{\mathcal{R}}(F(X)) \subseteq F(X)$.

Так как множество X конечно, то множество $F(X)$ тоже конечно. Тогда по указанной выше теореме множество всех $\mathcal{R}$-максимальных элементов и есть ядро бинарого отношения $\mathcal{R}$ на $F(X)$.

Для введенных понятий обобщенно близких вариантов и соответствующего бинарного отношения справедлива следующая теорема:

Теорема 4.1. Если бинарное отношение $\mathcal{R}$ соответствует тройке $(F(X), \mathcal{K}, y^*)$, то

$$C(F(X), \mathcal{K}, y^*) \subseteq \mathbf{K}_{\mathcal{R}}(F(X)).$$

Доказательство.

Пусть $A \in C(F(X), \mathcal{K}, y^*)$. Докажем утверждение теоремы от противного.

Предположим, что $A \notin \mathbf{K}_{\mathcal{R}}(F(X))$, тогда по определению ядра бинарного отношения существует $B \in F(X) : B\mathcal{R}A$. Следовательно, по определению $\mathcal{R}$ получим, что

$$\forall U(\cdot) \in \mathcal{K} \ \min U(B) > \max U(A).$$

Значит

$$\nexists U'(\cdot) \in \mathcal{K} : \max U'(A) \geqslant \min U'(A'), \forall A' \in F(X),$$

так как это не выполняется для $A' = B$. Отсюда получаем, что A не является обобщенно близким вариантом из $F(X)$ к цели y^* относительно класса функций $\mathcal{K}$, то есть $A \notin C(F(X), \mathcal{K}, y^*)$.

Получили противоречие. Значит, исходное предположение неверно, следовательно, $A \in \mathbf{K}_{\mathcal{R}}(F(X))$. Что и требовалось доказать.

Определение. Бинарное отношение $\mathcal{R}$, соответствующее тройке $(F(X), \mathcal{K}, y^*)$ назовем сильно соответствующим этой тройке, если

$$C(F(X), \mathcal{K}, y^*) = \mathbf{K}_{\mathcal{R}}(F(X)).$$

Получается, что задача поиска обобщенно близких вариантов сводится к построению сильно соответствующего бинарного отношения и выделению ядра этого бинарного отношения.

Пусть $\mathbf{K}_S(F(X))$ – ядро бинарного отношения доминирования по Слейтеру множества $F(X)$. Справедлива следующая

Теорема 4.2. Если $\mathcal{R}$ – соответствующее тройке $(F(X), \mathcal{K}, y^*)$ бинарное отношение, то

$$\mathbf{K}_{\mathcal{R}}(F(X)) \subseteq \mathbf{K}_S(F(X)).$$

Доказательство.

Для доказательства вложенности ядер двух бинарных отношений достаточно доказать, что

$$\forall A, B \in Q(\mathbb{R}^m) : A \succ_S B \text{ следует } A\mathcal{R}B.$$

Пусть $A \succ_S B$. Это значит, что $\forall a \in A, b \in B$ следует, что $a \succ_S b$.

По определению строго монотонных функций, если $a, b \in Y : a \succ_S b \Rightarrow U(a) > U(b)$. Значит, $\forall a \in A, b \in B \ U(a) > U(b)$. Следовательно, $\min U(A) > \max U(B)$.

В то же время бинарное отношение $\mathcal{R}$ соответствует тройке $(F(X), \mathcal{K}, y^*)$, если $ARB \Leftrightarrow \forall U(\cdot) \in \mathcal{K}$ имеет место $\min U(A) > \max U(B)$.

Значит, из $A \succ_S B$ следует $A\mathcal{R}B$. Что и требовалось доказать.

Следствие. $C(F(X), \mathcal{K}, y^*) \subseteq \mathbf{K}_S(F(X))$

Пусть $\mathbf{K}_P(F(X))$ – ядро бинарного отношения доминирования по Парето множества $F(X)$.

Поскольку $\mathbf{K}_P(F(X)) \subseteq \mathbf{K}_S(F(X))$, то возможно, что некоторые из элементов $C(F(X), \mathcal{K}, y^*)$ являются недоминируемыми по Парето, в то время как другие являются доминируемыми по Парето.

Стоит заметить, что для каждого доминируемого по Парето элемента из $C(F(X), \mathcal{K}, y^*)$ в этом множестве найдется доминирующий его элемент. Это следует из монотонности функций $U(\cdot)$. Пусть $A \succ B \Rightarrow$ для $\forall a \in A, b \in B, \forall U(\cdot) \in \mathcal{K}$ выполняется $U(a) \geqslant U(b) \Rightarrow \min U(A) \geqslant \max U(B)$. Пусть при этом $B \in C(F(X), \mathcal{K}, y^*)$. Это значит, что B является решением задачи (1) для какой-либо функции $U(\cdot) \in \mathcal{K}$. Тогда понятно, что и A будет являться решением для этой же задачи. Следовательно, $A \in C(F(X), \mathcal{K}, y^*)$.

Рассмотрим конкретный класс функций

$$\mathcal{K}_{Ch}^{y^*} = \{U(\cdot) | U(y) = -\max_i \lambda_i (y_i^* - y_i), \lambda_i > 0, \forall i\}.$$

Свойства класса $\mathcal{K}_{Ch}^{y^*}$:

1. Для $\forall U(\cdot) \in \mathcal{K}_{Ch}^{y^*}$ выполняется $U(y^*) = 0$.

   Доказательство.
   $U(y^*) = -\max\limits_i \lambda_i(y_i^* - y_i^*) = -\max\limits_i 0 = 0$, для $\forall U(\cdot) \in \mathcal{K}_{Ch}^{y^*}$.

2. $\max U(A) \leqslant 0$, для $\forall A \in F(X)$.

   Доказательство.
   Следует из того, что y^* принадлежит паретовой границе ВОЭП наилучших точек множеств $F(X)$. В этом случае получается, что для $\forall A \in F(X), \forall y \in A$ выполняется $y_i^* \geqslant y_i$. Тогда $U(y) = -\max\limits_i \lambda_i(y_i^* - y_i) \leqslant 0$.

3. $U(\cdot)$ — строго монотонные функции.

   Доказательство.
   Пусть $y \succ_S y' \Rightarrow$ для $\forall i$ $y_i > y_i' \Rightarrow y_i^* - y_i < y_i^* - y_i'$, для $\forall i \Rightarrow \max\limits_i \lambda_i(y_i^* - y_i) < \max\limits_i \lambda_i(y_i^* - y_i') \Rightarrow U(y) > U(y')$.
   Пусть $y \succ y' \Rightarrow$ для $\forall i$ $y_i \geqslant y_i' \Rightarrow y_i^* - y_i \leqslant y_i^* - y_i'$, для $\forall i \Rightarrow \max\limits_i \lambda_i(y_i^* - y_i) \leqslant \max\limits_i \lambda_i(y_i^* - y_i') \Rightarrow U(y) \geqslant U(y')$.

Из свойств $1-3$ класса функций $\mathcal{K}_{Ch}^{y^*}$ следует, что $\mathcal{K}_{Ch}^{y^*} \subset \mathcal{K}_M^{Y,y^*}$.
Определим бинарное отношение $\mathcal{R}_0$ на $F(X)$:

$$A\mathcal{R}_0 B \Leftrightarrow \text{для } \forall a \in A, \forall b \in B \text{ и } \forall i : a_i \leqslant y_i^* \text{ имеем } a_i > b_i.$$

Теорема 4.3. $\mathcal{R}_0$ **– соответствующее тройке** $(F(X), \mathcal{K}_{Ch}^{y^*}, y^*)$ **бинарное отношение.**
Доказательство.
По определению соответствия бинарного отношения надо доказать, что

$$\forall A, B \in F(X) : A\mathcal{R}_0 B \Leftrightarrow \min U(A) > \max U(B), \text{ для } \forall U(\cdot) \in \mathcal{K}_{Ch}^{y^*}.$$

Необходимость.
Пусть $A\mathcal{R}_0 B$. Докажем, что для $\forall U(\cdot) \in \mathcal{K}_{Ch}^{y^*}$ выполняется $\min U(A) > \max U(B)$. Берем $\forall a \in A, \forall b \in B$. Тогда для $\forall U(\cdot) \in \mathcal{K}_{Ch}^{y^*}$

$$U(a) = -\max_i \lambda_i(y_i^* - a_i), U(b) = -\max_i \lambda_i(y_i^* - b_i).$$

Так как $A\mathcal{R}_0 B$, то $b_i = a_i - c_i$, где

$$\begin{cases} c_i > 0, \text{при } a_i \leqslant y_i^*, \\ c_i \in (-\infty, +\infty), \text{при } a_i > y_i^*. \end{cases}$$

Рассмотрим следующее множество: $I = \{1, 2, ...m\}$. Разобьем I на два подмножества:

$$I_{\leqslant} = \{i \in I | a_i \leq y_i^*\},$$

$$I_{>} = \{i \in I | a_i > y_i^*\}.$$

Заметим, что $I_{\leq} \neq \emptyset$, так как y^* принадлежит паретовой границе ВОЭП наилучших точек исходных множеств.
Покажем, что $U(a) > U(b)$, то есть

$$-\max_i \lambda_i(y_i^* - a_i) > -\max_i \lambda_i(y_i^* - b_i).$$

В силу введенного $c = (c_1, c_2, \ldots, c_m)$ это тоже самое, что

$$\max_i \lambda_i(y_i^* - a_i + c_i) > \max_i \lambda_i(y_i^* - a_i).$$

Рассмотрим $\max\limits_i \lambda_i(y_i^* - a_i)$.

Максимум в данном случае достигается при $i \in I_{\leqslant} \neq \emptyset$. Это потому, что по построению множеств $I_>$ и $I_{\leqslant}$ при $i \in I_>$ получается, что $\lambda_i(y_i^* - a_i) < 0$, а при $i \in I_{\leqslant}$ имеем, что $\lambda_i(y_i^* - a_i) \geqslant 0$.

Рассмотрим $\max\limits_i \lambda_i(y_i^* - a_i + c_i)$.

Пусть $\max\limits_i \lambda_i(y_i^* - a_i + c_i)$ достигается при $i \in I_{\leqslant}$. Тогда так как $c_i > 0$, то $\max\limits_i \lambda_i(y_i^* - a_i + c_i) > \max\limits_i \lambda_i(y_i^* - a_i)$

Пусть $\max\limits_i \lambda_i(y_i^* - a_i + c_i)$ достигается при $i \in I_>$. Тогда

$$\max_{i \in I} \lambda_i(y_i^* - a_i + c_i) \geqslant \max_{i \in I_{\leqslant}} \lambda_i(y_i^* - a_i + c_i) >$$

$$> \{ \text{ так как } c_i > 0 \text{ для } i \in I_{\leqslant}\} > \max_{i \in I_{\leqslant}} \lambda_i(y_i^* - a_i) = \max_{i \in I} \lambda_i(y_i^* - a_i).$$

Таким образом

$$\max_i \lambda_i(y_i^* - a_i + c_i) > \max_i \lambda_i(y_i^* - a_i).$$

Значит, для $\forall U(\cdot) \in \mathcal{K}_{Ch}^{y^*}$

$$U(a) > U(b), \text{для } \forall a \in A, \forall b \in B.$$

Так как из-за компактности множеств A и B получается, что $\min U(A)$ и $\max U(B)$ достигаются. В силу произвольности точек $a \in A$ и $b \in B$ имеем, что

$$\min U(A) > \max U(B), \forall U(\cdot) \in \mathcal{K}_{Ch}^{y^*}.$$

Достаточность.

Пусть $\forall U(\cdot) \in \mathcal{K}_{Ch}^{y^*}$ выполняется $\min U(A) > \max U(B)$. Надо показать, что $A\mathcal{R}_0 B$. Докажем от противного.

Пусть $\overline{A\mathcal{R}_0 B}$. Тогда $\exists a \in A, \exists b \in B$: и $\exists j \in I : a_j < y_j^*$ и $b_j \geqslant a_j$. Построим $U'(\cdot) \in \mathcal{K}_{Ch}^{y^*}$: $U'(a) \leqslant U'(b)$. Тем самым будет получено противоречие.

$$U'(y) = \max_i \lambda_i'(y_i^* - y_i),$$

где

$$\lambda_i' = \begin{cases} 1, \text{при } i = j, \\ \varepsilon, \text{при } i \neq j, \end{cases}$$

ε – достаточно малое число.

Так как $y_j^* > a_j$, то $\exists \varepsilon$:

$$\begin{cases} \max\limits_i \lambda_i'(y_i^* - a_i) = y_j^* - a_j, \\ \max\limits_i \lambda_i'(y_i^* - b_i) = y_j^* - b_j. \end{cases}$$

Это значит, что

$$\max_i \lambda_i'(y_i^* - b_i) = y_j^* - b_j \leqslant y_j^* - a_j = \max_i \lambda_i'(y_i^* - a_i).$$

Отсюда следует, что

$$\max_i \lambda_i'(y_i^* - b_i) \leqslant \max_i \lambda_i'(y_i^* - a_i).$$

Следовательно

$$U'(a) \leqslant U'(b).$$

Получили противоречие, исходное предположение неверно, следовательно, $A\mathcal{R}_0B$.

Теорема полностью доказана.

Теорема 4.4. $\mathcal{R}_0$ **– сильно соответствующее тройке** $(F(X), \mathcal{K}_{Ch}^{y^*}, y^*)$ **бинарное отношение.**

Доказательство.

Необходимо доказать, что

$$C(F(X), \mathcal{K}_{Ch}^{y^*}, y^*) = \mathbf{K}_{\mathcal{R}_0}(F(X)).$$

Для этого докажем взаимное вложение:

1. $C(F(X), \mathcal{K}_{Ch}^{y^*}, y^*) \subseteq \mathbf{K}_{\mathcal{R}_0}(F(X))$,
2. $C(F(X), \mathcal{K}_{Ch}^{y^*}, y^*) \supseteq \mathbf{K}_{\mathcal{R}_0}(F(X))$.

Первое утверждение следует из теорем 4.1 и 4.3: в теореме 4.3 показано, что $\mathcal{R}_0$ – соответствующее тройке $(F(X), \mathcal{K}_{Ch}^{y^*}, y^*)$ бинарное отношение, а тогда по теореме 4.1 получается, что $C(F(X), \mathcal{K}_{Ch}^{y^*}, y^*) \subseteq \mathbf{K}_{\mathcal{R}_0}(F(X))$.

Докажем второе утверждение теоремы.

По сути, требуется доказать, что

$$\forall A \in \mathbf{K}_{\mathcal{R}_0}(F(X))\ \exists U'(\cdot) \in \mathcal{K}_{Ch}^{y^*} : \max U'(A) \geqslant \min U'(B), \forall B \in F(X).$$

Пусть $A \in \mathbf{K}_{\mathcal{R}_0}(F(X))$. Построим $U'(\cdot) \in \mathcal{K}_{Ch}^{y^*} : \max U'(A) \geq \min U'(B), \forall B \in F(X)$.

Пусть y' – наилучшая точка множества A, то есть точка с максимальными координатами среди всех точек множества A. Разобьем все множество индексов $I = \{1, 2, \ldots, m\}$ на подмножества $I_< = \{i \in I | y'_i < y^*_i\}$ и $I_\geqslant = \{i \in I | y'_i \geqslant y^*_i\}$.

Пусть $I_< = \emptyset$.

Тогда так как y^* принадлежит паретовой границе ВОЭП наилучших точек множества $F(X)$ получаем, что $y' = y^*$. Тогда $U(y') = 0$. Следовательно, по свойству 2 класса функций $\mathcal{K}_{Ch}^{y^*}$ получаем, что

$$\text{для } \forall U'(\cdot) \in \mathcal{K}_{Ch}^{y^*} : \max U'(A) \geqslant \min U'(B), \forall B \in F(X),$$

так как $\max U'(A) = U'(y') = 0$.

Пусть $I_< \neq \emptyset$.

Построим $U'(y) = -\max\limits_i \lambda'_i(y^*_i - y_i)$, где

$$\lambda'_i = \begin{cases} \dfrac{\varepsilon}{(y^*_i - y'_i)}, & \text{при } i \in I_<, \\ 1, & \text{при } i \in I_\geqslant. \end{cases}$$

Имеем

$$U'(y') = -\max_i(y^*_i - y'_i) = -\varepsilon.$$

А значит

$$\max U'(A) = U'(y') = -\varepsilon.$$

Предположим, что $\exists y \in Y : U'(y) > U'(y') = -\varepsilon$. Это то же самое, что и $\max\limits_i \lambda'_i(y^*_i - y_i) < \varepsilon$.

Пусть $\max\limits_i \lambda'_i(y^*_i - y_i)$ достигается при $i = j$. Возможны следующие варианты.

1. $j \in I_{\geqslant}$.
   Тогда $\lambda'_j = 1$ и так как ε достаточно мало, то

$$\max_i \lambda'_i(y^*_i - y_i) = \lambda'_j(y^*_j - y_j) = y^*_j - y_j > \varepsilon.$$

   Что противоречит предположению о том, что $U'(y) > U'(y') = -\varepsilon$.

2. $j \in I_{<}$.
   Тогда $\lambda_j = \frac{\varepsilon}{(y^*_j - y_j)}$.

$$\max_i \lambda'_i(y^*_i - y_i) = \lambda'_j(y^*_j - y_j) = \varepsilon\frac{y^*_j - y_j}{y^*_j - y'_j}.$$

   Если $\varepsilon\frac{y^*_j - y_j}{y^*_j - y'_j} < \varepsilon$, то $y^*_j - y_j < y^*_j - y'_j \Rightarrow y_j > y'_j$.

   Так как $\forall i\ \lambda'_i(y^*_i - y_i) < \varepsilon\frac{y^*_j - y_j}{y^*_j - y'_j}$, то $\forall i \in I_{<}\ \varepsilon\frac{y^*_i - y_i}{y^*_i - y'_i} < \varepsilon$ и тогда аналогично получаем, что $y_i > y'_i$.
   Заметим, что при $i \in I_{\geqslant}$ получаем, что $\lambda'_i = 1$. А значит, что при $y^*_i - y_i < \varepsilon$. получим $y_i > y^*_i$.

Таким образом, получаем, что если $U'(y) > U'(y')$, то $y_i > y'_i, \forall i \in I_{<}$ и $y_j > y^*_j, \forall j \in I_{\geqslant}$.

Пусть $\exists B \in F(X) : \max U'(A) < \min U'(B)$. Тогда в силу замкнутости множеств A и B получается, что

$$U'(y) > U'(y'), \forall y \in B, \forall y' \in A.$$

Тогда по доказанному получается, что

$$B\mathcal{R}_0 A.$$

Это противоречит предположению о том, что

$$A \in \mathbf{K}_{\mathcal{R}_0}(F(X)).$$

Значит,

$$\max U'(A) \geqslant \min U'(B), \forall B \in F(X).$$

Теорема полностью доказана.

Замечание. Доказанная терема показывает, что модифицированный МРЦ отбирает альтернативы критериальных параллелепипедов, на которых могут достигаться максимумы по хотя бы одной из функций $U(y)$ типа (4)относительно зафиксированной ЛПР целевой точки y^*, то есть МРЦ отбирает объединение всех решений задачи (1) на множестве $F(X)$ по всем функциям $U(\cdot) \in \mathcal{K}^{y^*}_{Ch}$. Более того, для каждой свертки из рассмотренного класса в полученной совокупности параллелепипедов все множества из $F(X)$, для которых этот максимум может достигаться.

Таким образом, решение бесконечного числа задач (1) сводится к выделению ядра некоторого бинарного отношения. По виду бинарного отношения $\mathcal{R}_0$ ясно, что для модифицированных параллелепипедов оно совпадает с бинарным отношением по Парето для элементов из $Q(\mathbb{R}^m)$.

5 Пример применения метода к конкретной задаче.

Описанная методика была применена к задаче улучшения качества воды в бассейне реки. Математическая модель, которая подробно описывает изучаемую проблему, наиболее полно представлена в работе [6]. Приведем краткое описание модели.

В модели предполагается, что задано K предприятий, осуществляющих выброс загрязненных сточных вод в реку. Концентрации загрязняющих веществ измеряются в гидрологических пунктах, расположенных ниже точек выброса.

Гидрологических пункты, расположенные ниже точек выброса, разбивают реку на K участков (створов). Расход воды Q_k в реке в районе k-го гидрологического пункта задан. Пусть I – число загрязнителей, рассматриваемых в модели. Уравнение баланса i-го загрязнителя для k-го створа имеет вид:

$$M_{ki} = \sum_{r=1}^{K} \delta_{ki}^{r} m_{ri} + \delta_{ki}^{0} m_i^0,$$

где
m_i^0 – сброс из источников, расположенных выше первого предприятия,
M_{ki} – поток i-го загрязнителя в реке через k-й гидрологический пункт,
δ_{ki}^{r} – коэффициент распада к k -му створу i-го загрязнителя, поступившего из r-го предприятия,
m_{ri} – сброс i-го загрязнителя в единицу времени в составе сточных вод r-го предприятия. Считается, что потоки m_i^0 каждого из I загрязнителей заданы заранее.

В технологической модели очистки стоков рассматривается совокупность из N возможных технологий очистки, заданных своими удельными приведенными затратами и коэффициентами очистки сбросов, различными для различных загрязнителей. Задача состоит в выборе некоторой технологии для каждого из K предприятий. Потоки загрязнителей в составе сточных вод после очистки, т.е. величины m_{ki}, заданы соотношениями:

$$m_{ki} = m_{ki}^0 (1 - \sum_{n=1}^{N} \delta_{in} t_{kn}), k = \overline{1, K}, i = \overline{1, I},$$

где
m_{ki}^0 – поток i-го загрязнителя в составе сточных вод, выбрасываемых k-м предприятием до строительства очистных сооружений,
δ_{in} – коэффициент очистки i-го загрязнителя при обработке сточных вод по n-й технологии очистки $(0 < \delta_{in} < 1)$,
t_{kn} – целочисленная переменная, равная единице, если технология применяется для очистки сточных вод k-го предприятия, и нулю в противном случае. Предполагается, что на каждом предприятии может быть использована только одна технология.

Концентрация i-го загрязнителя в речной воде в k-ой станции мониторинга вычисляется по формуле:

$$\varphi_{ki} = \frac{M_{ki}}{Q_k}, k = \overline{1, K}, i = \overline{1, I}.$$

Относительная концентрация i-го загрязнителя, измеряемая в используемых водохозяйственниками единицах предельно допустимой концентрации (ПДК), в k-м створе вычисляется по формуле:

$$Z_{ki} = \frac{\varphi_{ki}}{\varphi_i^{\max}},$$

где $\varphi_i^{\max}$ – ПДК для i-го загрязнителя. В качестве критериев загрязненности речной воды используются максимальные по всем створам значения относительных концентраций, т.е. величины

$$Z_i = \max_k Z_{ki}, i = \overline{1, I}.$$

Модель расчета затрат на очистку сточных вод имеет следующий вид. Поскольку каждая технология очистки описывается стоимостью обработки кубического метра стоков, то приведенные затраты на водоохранные мероприятия на k-м предприятии рассчитываются по формуле:

$$F_k = q_k \sum_{n=1}^{N} a_n t_{kn}, k = \overline{1, K}.$$

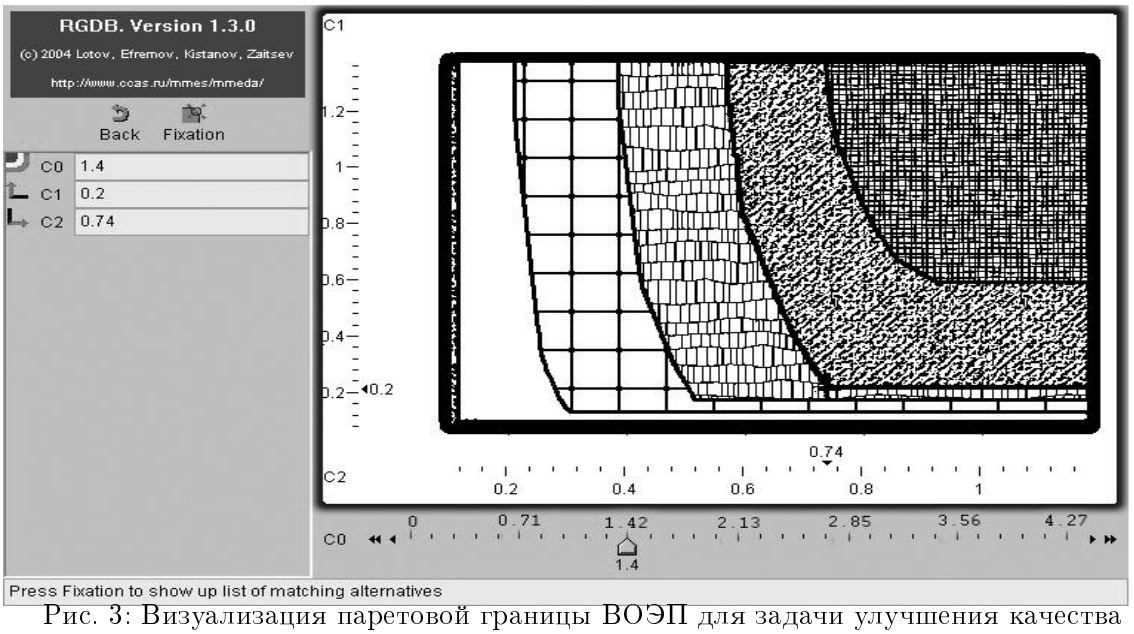


Рис. 3: Визуализация паретовой границы ВОЭП для задачи улучшения качества воды

где
q_k – объем стоков k-го предприятия в единицу времени,
a_n – стоимость обработки кубического метра стоков по n-й технологии очистки.

В качестве стоимостного критерия в системе используется общая стоимость проекта улучшения качества воды, равная сумме расходов по всем предприятиям.

Как уже говорилось, задача состоит в выборе технологий, которые будут использованы для очистки стоков для каждого из предприятий.

Для демонстрации применения методики, описанной с статье, нами было рассмотрено 3 критерия: стоимость проекта, а также максимальные по реке концентрации двух видов загрязнений. При этом рассматривалось 14 предприятий, где для каждого из предприятий считался возможным выбор одного из двух вариантов технологических решений по очистке выбросов. Таким образом, в нашем исследовании было рассмотрено $2^{14} = 16384$ возможных вариантов проекта очистки выбросов. Неопределенность задавалась отрезками возможного изменения одного из критериев, а именно стоимости проектов. В качестве минимального значения стоимости проекта бралась его стоимость в задаче с точными данными [7], далее задавалось процентное отклонение значения стоимости проекта от минимального значения (оно одинаково для всех проектов).

Например, пусть α_i – стоимость i-го проекта, ε – выбранное относительное отклонение (выраженное в процентах). Тогда значение стоимости i-го проекта будет лежать в отрезке $[\alpha_i, (1 + \frac{\varepsilon}{100})\alpha_i]$.

На рис.3 представлена черно-белая копия карты решений, представляющей паретову границу ВОЭП для этой задачи. Здесь по оси абсцисс показывается значение ПДК для второго типа загрязнения, а по оси ординат – ПДК первого вида загрязнения. Разным штриховкам соответствуют различные стоимости проектов. Карта решений построена с помощью системы RGDB [8].

Пусть в качестве целевой точки y^* пользователь (ЛПР) выбрал точку с координатами $(1.4, 0.2, 0.74)$. Первая координата - стоимость проекта в миллиардах рублей, вторая и третья координаты – ПДК первого и второго видов загрязнений соответственно.

В этом случае при помощи МРЦ в задаче без неопределенностей отбирается 4 альтернативы. Однако уже при малом отклонении 0.05% в стоимости проекта находятся 61 альтернатива.

Ниже приведена таблица, в которой представлена зависимость числа отобранных альтернатив от ε — относительного отклонения стоимости проектов от исходных значений.

Таблица результатов

Неопределенность	Число альтернатив
0%	4
0.05%	61
0.1%	82
0.5%	121
1%	133
2%	196

Из таблицы видно, что при наличии хотя бы небольшой неопределенности число отобранных альтернатив резко увеличивается. Дальнейшее увеличение неопределенности ведет к еще большему увеличению числа отобранных альтернатив, но в этом случае это число решений растет медленнее роста неопределенности. Таким образом, требуется разработка дополнительных средств, позволяющих уменьшить число отобранных альтернатив перед их детальным анализом.

Список литературы

[1] Д.В.Гусев, А.В. Лотов. *Методы поддержки принятия решений в задаче конечного выбора.* Исследование операций. Модели, системы, решения (под ред. Ю.П.Иванилова). М.: ВЦ РАН, 1994.

[2] A.V. Lotov. *Visualization-based Selection-aimed Data Mining with Fuzzy Data.* //International Journal of Information Technology & Decision Making. Vol. 5, No 4 (December 2006), 611-621.

[3] А.В. Лотов, А.В. Холмов. *Метод разумных целей в многокритериальных задачах с неточными данными.* Труды V Московской международной конференции по исследованию операций (ORM2007). М.: МАКС Пресс, 2007, 152-153.

[4] А.В. Лотов, В.А. Бушенков, Г.К. Каменев, О.Л. Черных. *Компьютер и поиск компромисса. Метод достижимых целей.* М.: Наука, 1997.

[5] А.В. Лотов, И.И. Поспелова. *Конспект лекций по теории и методам многокритериальной оптимизации.* М.: ВМК МГУ им. М.В. Ломоносова, 2006.

[6] Л.В. Бурмистрова, Р.В. Ефремов, А.В. Лотов. *Методика визуальной поддержки принятия решений и ее применение в системах управления водными ресурсами.* Известия Академии Наук. Серия Теория и Системы Управления. 2002, № 5, 89-100.

[7] A.V. Lotov, L.V. Bourmistrova, R.V. Efremov, V.A. Bushenkov, A.L. Buber, N.A. Brainin. *Experience of model integration and Pareto frontier visualization in the search for preferable water quality stratagies.* Environmental Modelling & Software 20, 2005, 243-260.

[8] A.V. Lotov, A.A. Kistanov, A.D. Zaitsev. *Visualization-based Data Mining Tool and its Web application.* In Y. Shi, W. Xu, and Z. Chen, editors, Data Mining and Knowledge Management. Chinese Academy of Sciences Symposium CASDMKD 2004, Beijing, China, July 12-14, 2004 series, Lecture Notes in Artificial Intelligence, volume 3327, pages 1-10, 2004. Springer-Verlag, Berlin.

УДК 519.688

ИССЛЕДОВАНИЕ ЗНАЧЕНИЙ ПОКАЗАТЕЛЯ СЕМАНТИЧЕСКОЙ СОВМЕСТИМОСТИ ДЛЯ ПАР СЛОВ ПРИЛАГАТЕЛЬНОЕ–СУЩЕСТВИТЕЛЬНОЕ

© 2008 г. А. П. Котляров

koterpillar@gmail.com

Кафедра Алгоритмических языков

1 Введение

Знания о допустимости в текстах на естественном языке тех или иных словосочетаний важны для многих прикладных задач, связанных с машинным пониманием и генерацией текста на естественном языке. Для решения этих задач нужна информация о том, какие слова сочетаются или не сочетаются друг с другом, а также о том, насколько смысл полученных словосочетаний отличается от смысла компонент. Допустимость словосочетаний обычно определяется путем статистического анализа текстовых корпусов и последующей оценки экспертом. Отобранные словосочетания хранятся в формализованном виде в компьютерных словарях словосочетаний.

Подобный словарь является ядром системы КроссЛексика [1] — многофункционального тезауруса, наполнение которого производит эксперт-лингвист на основе просмотра множества текстов. КроссЛексика содержит допустимые в русском языке словосочетания, разделенные по типам (стрелкой показана синтаксическая зависимость): существительное–существительное (например, *полоса прибоя*, *коробка с карандашами*), существительное–глагол (*пить чай*, *думать о делах*), прилагательное–существительное (*цветное стекло*, *бегущий человек*) и другие. Однако в словаре КроссЛексики могут отсутствовать словосочетания, появившиеся сравнительно недавно.

Новые словосочетания быстро отражаются в текстах сети Интернет, и поисковые машины, индексирующие страницы Интернета, дают возможность рассматривать его как текстовый корпус [2]. Несмотря на наличие шума, использование Интернета в качестве корпуса позволило успешно решить задачу об определении смысловой близости существительных на основе сочетающихся с ними прилагательных [3], а также об автоматизированном обнаружении малапропизмов (лексико-семантических ошибок, заключающихся в замене одного слова на другое, похожее по написанию и звучанию) [4].

При решении этих задач для оценки допустимости словосочетаний, т.е. совместимости между собой составляющих их слов, была введена мера семантической совместимости SCI (Semantic Compatibility Index). SCI вычисляется по возвращаемому поисковой машиной количеству встреч в Интернете слов вместе и по отдельности, измеренному в веб-страницах. Чем чаще исследуемые слова встречаются вместе, тем большее значение принимает SCI, поэтому для разделения словосочетаний по допустимости использовалось пороговое правило вида $SCI > T$, где константа T подбиралась для каждой задачи экспериментально.

Показатель SCI и пороговое правило-критерий допустимости словосочетаний построены по аналогии с показателем взаимной информации MII [5], используемом при анализе текстовых корпусов. Важными особенностями SCI являются независимость от постоянного роста массивов индексированных страниц, а также то, что для его вычисления не требуется знание размера массива текстов, в которых ищутся слова.

Хотя проведенные эксперименты [3, 4] доказали применимость показателя SCI для задач, где требуется разделить словосочетания по их семантической допустимости, остается необходимость исследования его как меры сочетаемости слов на более широком множестве словосочетаний, в том числе фразеологизмов — словосочетаний, смысл которых не выводится напрямую из смысла составляющих их слов [6]. В данной работе описывается эксперимент, в котором рассматриваются и оцениваются словосочетания типа прилагательное–существительное, взятые

из словаря КроссЛексики и нескольких известных словарей фразеологизмов. Также анализировались специально построенные пары слов того же синтаксического типа, для которых заранее не была известна их семантическая совместимость. Общее число рассмотренных словосочетаний составляет 69081 — почти на порядок больше, чем в предыдущих экспериментах.

2 Меры оценки сочетаемости

Для поиска устойчивых двусловных словосочетаний в корпусах текстов используется показатель взаимной информации MII (Mutual Information Index) [7, 5]:

$$MII = \log_2 \frac{S \cdot N_{12}}{N_1 \cdot N_2}$$

Здесь N_{12} — количество употреблений в корпусе самого словосочетания, N_1 и N_2 — количество встреч в корпусе составляющих его слов, а S - размер корпуса. Эта статистическая мера определяет, насколько часто употребляется словосочетание по сравнению с независимым употреблением своих компонент.

Мера MII не может быть применена для поиска устойчивых словосочетаний в Интернете без модификации, так как поисковые службы Интернета не выдают количество встреч слова или слов во все индексированные тексты, а лишь количество страниц, на которых они встречаются (возможно, по несколько раз на одной и той же странице). К тому же полный объем текстов, индексированных поисковой машиной, неизвестен, и, хотя можно предпринять шаги для его оценки [2], она быстро устаревает из-за постоянного роста массивов текстов Интернета.

Для оценки устойчивости и допустимости двусловных словосочетаний в [4] была введена модифицированная мера — показатель семантической совместимости SCI (Semantic Compatibility Index):

$$SCI = \log_2 \frac{P \cdot N_{12}}{\sqrt{N_1 \cdot N_2}}$$

Безразмерная константа $P = 2^{16}$ была подобрана так, чтобы значение SCI для допустимых словосочетаний оставалось положительным. Введение в знаменатель квадратного корня обеспечивает независимость значений SCI от равномерного увеличения величин N, измеряемых теперь в страницах.

Отметим, что если словосочетание вообще не встречается в корпусе или в Интернете ($N_{12} = 0$), значения MII и SCI для него принимаются равным $-\infty$.

3 Оценка допустимости пар слов

В эксперименте по оценке семантической совместимости слов участвовали три группы пар слов вида прилагательное–существительное: группа словосочетаний из словаря системы КроссЛексика, специально построенные с помощью того же словаря пары, и группа фразеологизмов, выбранных из нескольких словарей.

Для первой группы из словаря КроссЛексики были выбраны 30 существительных, обозначающих как конкретные, так и абстрактные понятия, и для каждого взято все множество его определений (прилагательных), представленных в словаре. Таких словосочетаний оказалось 4441. В среднем на одно существительное приходилось около 160 прилагательных (максимум — 516 у слова *глаза*, минимум — 10 у слова *пояснение*).

Вторую группу составили «перекрестные» пары слов, построенные следующим образом. К каждому из 30 взятых существительных присоединялись прилагательные, являющиеся определениями для какого-либо из 29 других, но не указанные как определения к нему самому (см. рисунок 1). Всего таких пар оказалось 64397. Для них, в отличие от ранее выбранных напрямую из КроссЛексики пар, заранее не было известно ничего об их семантической совместимости. Некоторые прилагательные являлись определениями сразу для нескольких выбранных существительных (чаще всего встречалось определение *хороший*). Они не включались в перекрестные пары ни с одним из этих существительных.

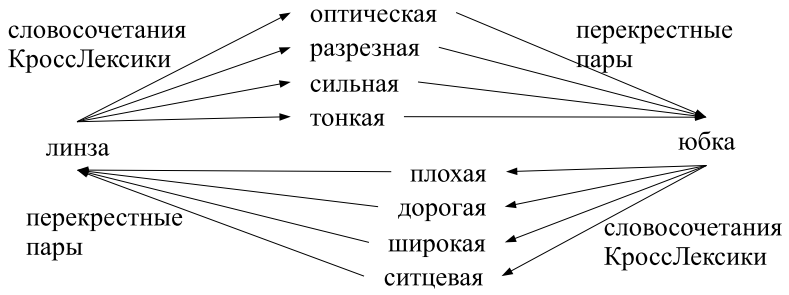


Рис. 1: Пример построения пар прилагательное–существительное

	КроссЛексика	Фразеологизмы	Перекрестные пары
Общее число пар	4441	243	64397
Среднее значение *SCI*	3,73	8,06	−3,21
Дисперсия *SCI*	3,29	2,87	3,97
Минимум *SCI*	−8,8	−0,28	−13,29
Максимум *SCI*	14,85	14,37	12,2
Не найдены в Интернете	120 (3%)	0	24771 (38%)
Пары со SCI $>$ 4	2179 (49%)	222 (91%)	1673 (3%)
Пары со SCI $>$ 7	642 (14%)	166 (68%)	179 (0,28%)

Таблица 1: Статистика *SCI* для исследуемых пар слов

Из нескольких фразеологических словарей была выбрана третья группа из 243 словосочетаний того же вида прилагательное–существительное, например, *новый русский*, *белая ворона*.

Для запросов к Интернету использовалась русскоязычная поисковая служба Яндекс [8]. Для каждой исследуемой пары прилагательное–существительное был произведен запрос к Яндексу и рассчитан показатель семантической совместимости. Распределение значений *SCI* для всех построенных пар показано в таблице 1, а его график представлен на рисунке 2. Пары, не найденные в Интернете вообще, в подсчет среднего значения и дисперсии не включались.

В среднем самую высокую оценку *SCI* ожидаемо получили фразеологизмы, а самую низкую — перекрестные пары. В то же время у перекрестных пар самая высокая дисперсия — среди них оказалось довольно много как недопустимых, так и весьма устойчивых словосочетаний.

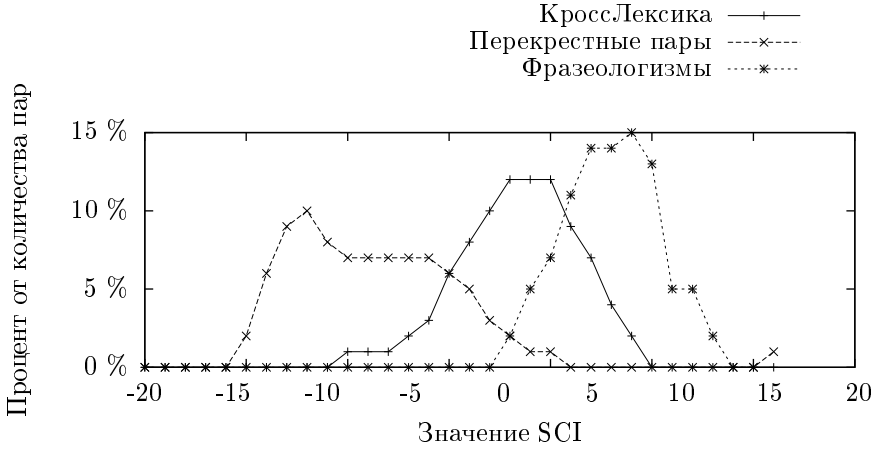


Рис. 2: Распределение значений *SCI* для исследуемых пар

	Недопустимые	Допустимые
Пары КроссЛексики	7	193
Перекрестные пары	146	54
Перекрестные пары с $4 < SCI < 7$	11	89
Перекрестные пары со $SCI > 7$	7	93

Таблица 2: Результаты оценки словосочетаний экспертом

Пары	Недопустимые	Допустимые
Общее число	153	247
Среднее значение SCI	-4.87	2.77
Дисперсия SCI	3.09	3.71
Минимум SCI	-10.23	-9.43
Максимум SCI	3.6	9.36
Не найдены в Интернете	74	12

Таблица 3: Оценки SCI для групп, построенных экспертом

4 Экспертная оценка пар слов

Для проверки корректности SCI как меры семантической допустимости из всего множества оцененных были взяты 600 пар: 200 словосочетаний из словаря КроссЛексики, 200 перекрестных пар и по 100 отдельно выбранных перекрестных пар, для которых SCI лежал в диапазонах соответственно $[4;7]$ и $[7;+\infty)$. Все пары выбиралась случайным образом из соответствующего множества.

Выбранные пары были отданы на независимую оценку эксперту, который разделял пары на два множества: допустимые и недопустимые в речи. Результаты оценки показаны в таблице 2, а статистика значений SCI среди всех пар — в таблице 3.

Экспертная оценка показала, что, несмотря на обилие недопустимых среди перекрестных пар, допустимые пары встречаются довольно часто. Среди пар, получивших оценку $SCI > 4$, 91% являются допустимыми. Увеличение порога с 4 до 7 не дает существенного прироста доли допустимых пар, причем она близка к доле допустимых словосочетаний, взятых собственно из словаря КроссЛексики, где допустимыми были признаны 96,5% (недопустимость оставшихся 3,5% можно рассматривать как неточность экспертной оценки).

Среднее значение SCI среди допустимых по экспертной оценке пар гораздо выше, чем для недопустимых, однако некоторые допустимые пары все же получают большую по абсолютной величине отрицательную оценку (например, $-8,8$ для пары *недорогая шерсть* и $-6,34$ для пары *осмысленная интонация*). Этот факт, а также 12 допустимых пар, не найденных вообще, таких, как *чайнонарезные автоматы* и *неугодное объяснение*, свидетельствуют о неполноте массивов русскоязычного Интернета.

5 Заключение

Проведенные эксперименты подтвердили, что мера семантической совместимости, введенная ранее, может применяться для определения семантической допустимости словосочетаний на основе массивов Интернета, и корректность этого применения подтверждена экспертной проверкой.

Способ построения перекрестных пар слов, а также сами построенные в ходе эксперимента перекрестные пары, получившие высокую оценку SCI, могут также быть использованы для автоматизированного пополнения словаря системы КроссЛексика и других баз словосочетаний.

Благодарности

Работа выполнена при финансовой поддержке РФФИ (проект 06-01-00571). Автор также выражает признательность И.А. Большакову за предоставление доступа к системе КроссЛексика.

Список литературы

[1] *Большаков И. А.* Многофункциональный словарь-тезаурус для автоматизированной подготовки русских текстов // *НТИ, серия 2.* — 1994. — №. 1. — Стр. 11–23.

[2] *Kilgarriff A., Grefenstette G.* Introduction to the special issue on the web as corpus // Computational linguistics. — Сер. 29. — 2003. — Стр. 333–347.

[3] *Bolshakov I. A., Gelbukh A. F.* Distribution-based semantic similarity of nouns // CIARP / Ed. by L. Rueda, D. Mery, J. Kittler. — Сер. 4756 *Lecture Notes in Computer Science.* — Springer, 2007. — Стр. 704–713.

[4] *Большакова Е., Большаков И., Котляров А.* Расширенный эксперимент по автоматическому обнаружению и исправлению русских малапропизмов // Диалог-2006. Компьютерная лингвистика и интеллектуальные технологии. — 2006. — Стр. 78–84.

[5] *Church K. W., Hanks P.* Word association norms, mutual information, and lexicography // Proceedings of the 27th. Annual Meeting of the Association for Computational Linguistics. — Vancouver, B.C.: Association for Computational Linguistics, 1989. — Стр. 76–83.

[6] *Розенталь Д. Э., Голуб И. Б., Теленкова М. А.* Современный русский язык. — М.: Айрис-Пресс, 2002.

[7] *Manning C. D., Schütze H.* Foundations of Statistical Natural Language Processing. — Cambridge, Massachusetts: The MIT Press, 1999. citeseer.ist.psu.edu/635422.html.

[8] Яндекс.XML: документация, 2008. http://help.yandex.ru/xml/?id=396706.

УДК 517.956.2

СТРУКТУРНО НЕУСТОЙЧИВЫЕ ВЕКТОРНЫЕ КОНФИГУРАЦИИ В УРБАНИСТИКЕ

© 2008 г. И. А. Куров

ivankur@mail.ru

Кафедра Автоматизации научных исследований

1 Введение

Современная наука, прежде всего, основывается на системном подходе к изучению явлений и процессов, происходящих при развитии различных систем. В настоящее время все чаще встречается такое понятие, как самоорганизация открытой системы. То есть система развивается по некоторым своим внутренним законам под воздействием внешних влияний или воздействий, приспосабливаясь за счет определенных внутренних ресурсов к существованию при условиях внешней среды.

При рассмотрении экономической активности на определенном географическом пространстве, значение имеет не только географическое положение различных объектов, но также система расселения жителей на этой территории, расположение рекреационных зон и производственной инфраструктуры, очень большое значение имеют, кроме того, транспортные сети и их пропускные способности на данной территории. По сути, все указанные факторы формируют единую сферу жизнедеятельности человеческого общества, которая развивается, учитывая каждый из факторов в рамках единой социально-производственной системы.

При рассмотрении и анализе каждой отдельно выбранной территории необходимо особое внимание уделять степени освоения данной местности, учитывать уровень развития транспортной системы. В силу неоднородности социально-экономического развития территорий приходится искать пути оптимального и наиболее выгодного развития производственной, транспортной и социальной инфраструктуры.

При анализе и моделировании развития отдельных территорий крайне интересным является вопрос о сохранении характеристик и свойств рассматриваемой территории, как социально-производственной системы при изменениях различных влияющих на нее факторов. Другими словами - вопрос о структурной устойчивости моделируемой системы.

Математические аспекты структурной устойчивости обсуждаются в монографии [1], которая дает следующее определение: система является структурно устойчивой, если для всякого достаточно малого изменения векторного поля, получаемая система эквивалентна исходной. Поэтому, естественно ожидать, что структурно неустойчивые конфигурации могут быть преобразованы в структурно устойчивые.

Мы рассматриваем эволюцию урбанистической системы как аналог задачи структурной устойчивости векторных полей. Экономическая активность характеризуется определенным временем и местом, поэтому разумно учитывать не только временные, но и пространственные зависимости при расчетах в моделировании нелинейного развития экономических систем. Под воздействием внешних воздействий, или в силу структурной неустойчивости в результате самосогласованной эволюции могут происходить изменения в топологии урбанистической системы.

Урбанистическая система рассматривается как структурно-неустойчивое азимутальное симметричное векторное поле с одной критической точкой высокого порядка. На основе компьютерного моделирования мы исследовали динамику развития городских центров и продемонстрировали различные городские системы, которые могут существовать: одноядерные системы, двуядерные системы и мульти-ядерные системы. Возможны различные варианты расположения больших городов в одноядерной системе:

- Центральное расположение столицы, как в Испании;

- Смещенное расположение столицы, как в Австрии;
- Периферийное расположение больших городов, как в Азии.

Аналогичные процессы имеют место и для современных мегаполисов, где бизнес центр играет роль ядра.

2 Базисная урбанистическая модель

При построении урбанистической модели развития города использовалась модель предложенная Бекманом, описанная в книге Пу. Модель Бекмана основывается на двух уравнениях, в частных производных, выражаемых в градиентном и дивергентном законах [2]. Градиентный закон выражается уравнением

$$k\frac{\phi}{|\phi|} = \nabla\lambda, \tag{1}$$

где k - локальная стоимость перевозки товара в данной точке, ϕ является вектором локального потока товаров, взятого в направлении $(\sin\phi, \cos\phi) = \frac{\phi}{|\phi|}$, ϕ - количество перемечаемого товара, а λ обозначает цену товара. Все указанные величины являются функциями пространственных переменных x, y и времени t. Это уравнение показывает, что любой поток имеет направление в сторону максимального роста цены товара и что цена товара зависит от стоимости перевозки. Из уравнения (1) следует, что зависимость между локальной стоимостью перевозки товара и ценой товара выражается

$$k = |\nabla\lambda|. \tag{2}$$

Тогда из уравнений (1) и (2) мы имеем $\phi = -\chi\nabla\lambda$, где $\chi(\lambda)$ - некоторая скалярная функция λ аналогичная коэффициенту теплопроводности. Дивергентный закон в модели Бекмана имеет вид:

$$div\phi + z(\lambda) = 0, \tag{4}$$

где $z(\lambda)$ - избыток предложения. В нестационарном случае из градиентного и дивергентного закона, данных в уравнениях (1) и (3) мы получаем уравнение для λ

$$\frac{\partial\lambda}{\partial t} + div(\chi\nabla\lambda) = -z(\lambda), \tag{4}$$

из которого получаем фундаментальный закон сохранения

$$\frac{\partial A}{\partial t} + div\Phi = F(A), \tag{5}$$

где A - плотность некоторой величины, Φ - вектор течения плотности и $F(A)$ источник мощности. В случае если $\Phi = AV$ уравнение описывает конвективный перенос, а в случае $\Phi = -k\nabla A$ описывает рассеивание.

3 Результаты компьютерного моделирования

Мы представляем результаты серии вычислений самосогласованного развития векторной урбанистической конфигураций.

Расчетная область представляла собой шестиугольник. Выбор именно такой расчетной области основан на классической теории центральных мест Кристаллера, именно такое районирование позволяет достигнуть максимальной площади замощения.

Основной постулат теории центральных мест заключается в том, что размещение экономической деятельности главным образом определяется условиями спроса и предложения. Предполагается также транспортная доступность во всех направлениях. На такой территории издержки снабжения поселения будут зависеть только от расстояния между местом производства товара и этим поселением. С увеличением издержек спрос на большинство товаров

уменьшается, и поэтому очевидно, что с ростом расстояния спрос на любой товар в любом районе будет уменьшаться. А так как население, в свою очередь, размещено равномерно и транспортные издержки пропорциональны расстоянию, то зона сбыта любого товара будет иметь форму круга и место производства этого товара расположится в центре зоны сбыта, то есть станет "центральным местом", а все поселения, которые снабжаются из этого центра, будут "зависимыми"местами. Появляется понятие конус спроса - радиус зоны сбыта центральных товаров, нижний предел которого определяется пороговым размером рынка, а верхний - расстоянием, вне которого центральное место уже неспособно сбывать свой товар.

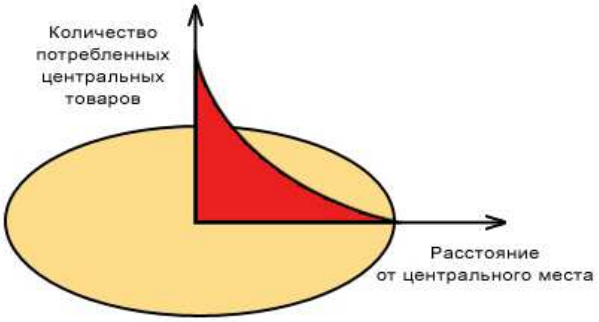


Рисунок 1. Модель конус спроса

Весь изучаемый район можно было бы разделить на ряд зон круглой формы, определяемых конусом спроса, но здесь возникает определенная трудность: если окружности касаются друг друга, то возникают необслуживаемые территории, если же окружности наоборот заполняю всю территорию, то они должны пересекаться вследствие чего возникают зоны перекрытия. Поэтому наиболее эффективной формой районов сбыта является форма правильного шестиугольника. Шестиугольная (гексагональная) структура возникает в результате стремления разместить на плоскости максимально возможное количество конусов спроса. Районы в форме шестиугольника равномерно заполняют всю территорию.

Кристаллер сформулировал выявленные закономерности следующим образом: группа тождественных центральных мест имеет шестиугольные дополняющие районы, а сами центральные места образуют правильную треугольную решетку. Размещение городов в модели Кристаллера обеспечивает оптимальное перемещение потребителей товаров и услуг - к самым близким к месту их проживания центральным местам. Таким образом рыночная, транспортная инфраструктура и административная структура оптимизируются.

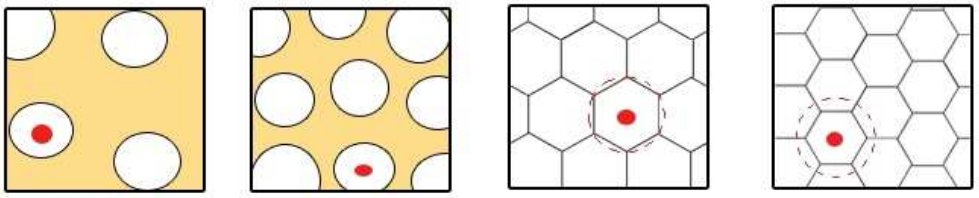


Рисунок 2. Примеры замощения территории

На основе предыдущего раздела нами было получено уравнение, характеризующее динамическое развитие урбанистической системы:

$$\frac{\partial \lambda}{\partial t} + div(\chi(\lambda) grad \lambda) = -z(\lambda), \tag{6}$$

Как уже было указано, здесь $\lambda(x, y, t) > 0$ - локальная стоимость товара, χ - коэффициент диффузии, $z(\lambda)$ локальный избыток предложения. Нами рассматривались системы, с нулевым локальным избытком предложения, и граничными условиями, соответствующими оптимальным

торговым условиям между взаимодействующими регионами города, то есть $\lambda = \lambda_0 + \frac{0{,}5}{\sqrt{x^2+y^2}}$, где $\lambda_0 = \lambda(x, y, t = 0)$ - начальное состояние системы.

При расчетах нами была использована явная разностная схема с сеткой 200x200.

В качестве начального состояния системы мы рассматривали урбанистическую векторную конфигурацию с осевой симметрией третьего порядка. Такая конфигурация задается следующим образом: $\lambda(x, y, t = 0) = x^3 - 3xy^2$.

При подобном моделировании урбанистической системы аналогами дорог являются "силовые линии"в получаемой векторной модели.

На рисунках 3-4 схематично показан процесс эволюции системы во времени. Отчетливо видно, что с течением времени происходит "развал"одного центра на три дополнительных, при этом следует отметить, что центральный сохраняется, меняется топология дорог.

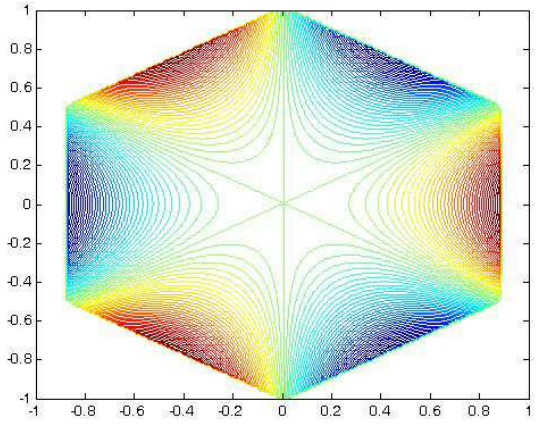


Рисунок 3. Начальное состояние моделируемой урбанистической системы в момент времени $t = 0$.

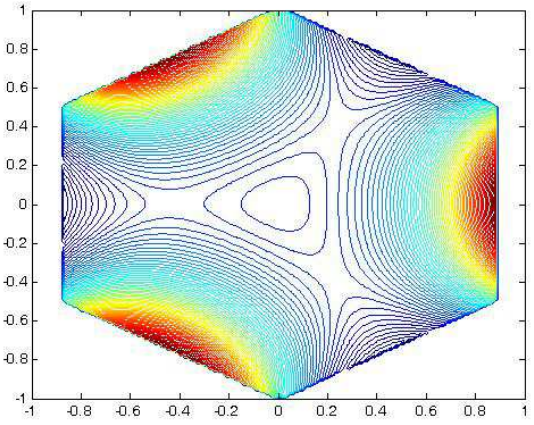


Рисунок 4. Состояние моделируемой урбанистической системы в момент времени $t = 4$.

Рассматриваемая модельная конфигурация имеет множество подтверждающих примеров в виде структуры различных городов, один из которых приведен на рисунке 5.

Рисунок 5. Пример города с конфигурацией дорог, подобных рисунку 1.

Во второй серии вычислений была использована та же модель, с теми же граничными условиями, а в качестве изначально заданной урбанистической векторной конфигурации была

выбрана следующая: $\lambda(x, y, t = 0) = x^3 - 3xy^2 + 0,5x$.

Рисунки 6-7 схематично показывают, что будет происходить с подобной системой в процессе самосогласованного развития. В этом случае система с двумя соединенными центрами, с течением времени трансформируется в систему с двумя самостоятельно существующими центрами.

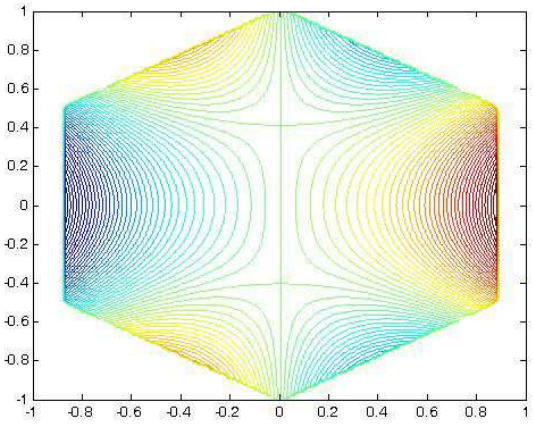


Рисунок 6. Начальное состояние моделируемой урбанистической системы в момент времени $t = 0$.

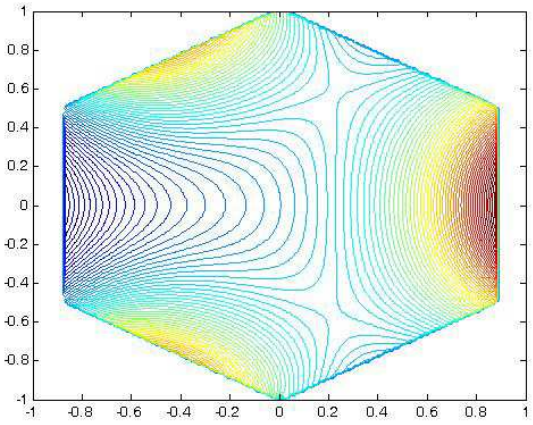


Рисунок 7. Состояние моделируемой урбанистической системы в момент времени $t = 4$.

Примерами городов с подобной структурой служат Канзас и Болонья.

Рисунок 8. Структура города Болоньи.

4 Заключение

В этой статье представлены результаты математического моделирования развития различных урбанистических систем, и было показано, что урбанистическая система стремится к

структурно устойчивым конфигурациям, вне зависимости от начальных и граничных условий, которые влияют только на необходимое для перестроения время. Проведение подобных исследований поможет проведению исследования развития городских структур и прогнозированию изменения их структуры, что является крайне важным в современном мире.

Список литературы

[1] В.И. Арнольд, Дополнительные главы теории обыкновенных дифференциальных уравнений, Наука, (1978)

[2] Т. Пу, Нелинейная экономическая динамика, издательский дом "Ижевский университет"(2000)

[3] В.-Б. Занг, Синергетическая экономика время и перемены в нелинейной экономической теории, Издательство Мир, (1999)

УДК 517.977

ПРИНЦИП МАКСИМУМА ПОНТРЯГИНА ДЛЯ НЕАВТОНОМНЫХ МОНОТОННЫХ ЗАДАЧ ОПТИМАЛЬНОГО УПРАВЛЕНИЯ НА БЕСКОНЕЧНОМ ИНТЕРВАЛЕ ВРЕМЕНИ.

© 2008 г. Н. А. Малыш

nadja@ldis.cs.msu.ru

Кафедра Оптимального Управления

1 Постановка задачи

Рассматривается неавтономная задача оптимального управления с интегральным функционалом на бесконечном интервале времени (P):

$$\dot{x}(t) = f(x(t), t, u(t)), \quad u(t) \in U, \quad t \geq 0; \tag{1}$$

$$x(0) = x_0 \tag{2}$$

$$J(x, u) = \int_0^\infty f^0(x(t), t, u(t))dt \to \max,$$

где $x(t) = (x^1(t), \ldots, x^n(t)) \in \mathbb{R}^n$, - значение фазового вектора x, а $u(t) = (u^1(t), \ldots, u^m(t)) \in \mathbb{R}^m$ - значение управления u в момент времени $t \geq 0$. Класс допустимых управлений состоит из измеримых функций $u : [0, \infty) \to \mathbb{R}^m$ со значениями из непустого компакта U. Векторная функция f и скалярная функция f^0 непрерывны вместе со своими частными производными по аргументам x и t на множестве $G \times [0, \infty) \times U$, где G - такое заданное открытое множество в $\mathbb{R}^n$, что $x_0 \in G$.

В данной работе рассматривается случай, когда функция f^0 имеет специальный вид

$$f^0(x, t, u) = e^{-\rho t} g(x, u) \qquad \text{для всех} \quad x \in G, u \in U, t \geq 0, \tag{3}$$

где $\rho > 0$ - параметр дисконтирования, а функция g непрерывна вместе со своей частной производной по переменной x на множестве $G \times U$. Задачи данного типа часто возникают в экономике.

Предполагается, что для любого допустимого управления u, соответствующее ему решение (Каратеодори) x системы (1), удовлетворяющее начальному условию (2), определено на $[0, \infty)$ и принимает значения в G.

С математической точки зрения, основная трудность исследования рассматриваемой задачи связана с бесконечным интервалом времени. Наличие бесконечного интервала вносит в задачу особенность, что является причиной различного рода патологий в соотношениях принципа максимума Понтрягина, в частности, в условиях трансверсальности на бесконечности (см. [1],[5],[9]).

При этом, в отличие от задач на конечном интервале времени, в случае задач на бесконечном интервале, неавтономный случай не сводится к автономному посредством введения дополнительной фазовой координаты $x^{n+1} \equiv t$, а требует специального рассмотрения. Это связано с тем обстоятельством, что при осуществлении этого приема используется условие трансверсальности в правом конце (см. [3]), которое в задачах на бесконечном интервале времени, вообще говоря, отсутствует.

Дальнейшие рассуждения проводятся при выполнении следующих условий:

Н 1 *Существует такая непрерывная на* $[0, \infty)$ *функция* $C(t) \geq 0$, *что*

$$\langle x, f(x, t, u) \rangle \leq C(t)(1 + \|x\|^2) \quad \textit{для всех } x \in G, u \in U, t \geq 0.$$

Н 2 *Для любых $t \geq 0, x \in G$ функция $u \to f(x,t,u)$ является аффинной по переменной u, т.е.*

$$f(x,t,u) = f_0(x,t) + \sum_{i=1}^{m} f_i(x,t)u^i \quad \textit{для всех } x \in G, u \in U, t \geq 0,$$

где все функции f_i, $i = 1 \dots m$ непрерывно дифференцируемы по совокупности переменных x, t.

Н 3 *Множество U выпукло. Для любых фиксированных $t > 0$, $x \in G$ функция $u \to g(x,u)$ вогнута по переменной u.*

Н 4 *Существуют такие невозрастающие на $[0,\infty)$ функции $\mu(t) \geq 0$, $\omega(t) \geq 0$, что $\lim_{t\to\infty} \mu(t) = 0$, $\lim_{t\to\infty} \omega(t) = 0$ и для любой допустимой пары (x,u), выполняются следующие условия:*

$$\max_{u\in U} e^{-\rho t}|g(x(t),u(t))| \leq \mu(t) \quad \textit{при всех } t > 0;$$

$$\int_T^\infty e^{-\rho t}|g(x(t),u(t))|dt \leq \omega(T) \quad \textit{при всех } T > 0.$$

В силу условий (Н1)-(Н4) оптимальная допустимая пара (x_*, u_*) в задаче (P) существует [1],[6].

2 Построение аппроксимирующей последовательности задач

Построим последовательность задач $\{(P_k)\}$, $k = 1, 2, \dots$ на конечных интервалах времени $[0, T_k]$ при $T_k \to \infty$ при $k \to \infty$, аппроксимирующую исходную задачу (P), используя подход, аналогичный изложенному в работах [1], [5].

Пусть (x_*, u_*) - оптимальная пара в исходной задаче (P). В силу ограниченности множества U существуют последовательности гладких функций $\{z_k\}, k = 1, 2, \dots$ и чисел $\{\sigma_k\}, k = 1, 2, \dots$, удовлетворяющие соотношениям

$$\sup_{t\in[0,\infty)} \|z_k(t)\| \leq \max_{u\in U} \|u\| + 1; \tag{4}$$

$$\int_0^\infty e^{-t}\|z_k(t) - u_*(t)\|^2 dt \leq \frac{1}{k}; \tag{5}$$

$$\sup_{t\in[0,\infty)} \|\dot{z}_k(t)\| \leq \sigma_k < \infty, \quad \lim_{k\to\infty} \sigma_k = \infty. \tag{6}$$

Для построенных последовательностей $\{z_k\}$ и $\{\sigma_k\}$,выберем последовательность положительных чисел $\{T_k\}$, $T_k < T_{k+1}$, $k = 1, 2, \dots$, $\lim\limits_{k\to\infty} T_k = \infty$ так, чтобы выполнялось неравенство

$$\omega(T_k) \leq \frac{1}{k(1+\sigma_k)} \quad \text{при всех } k = 1, 2, \dots.$$

Теперь определим аппроксимирующую последовательность задач $\{(P_k)\}, k = 1, 2, \dots$.

$$\dot{x}(t) = f(x(t), t, u(t)), \quad u(t) \in U \quad t \geq 0; \tag{7}$$

$$x(0) = x_0; \tag{8}$$

$$J_k(x,t,u) = \int_0^{T_k} \left[e^{-\rho t} g(x(t),u(t)) - \frac{1}{1+\sigma_k} e^{-t}\|z_k(t) - u(t)\|^2 \right] dt \to \max_{u\in U}.$$

В силу [4] для любого $k = 1, 2, \dots$ в задаче $\{(P_k)\}$ существует оптимальное управление u_k. Обозначим через x_k - допустимую траекторию системы (7), удовлетворяющую начальному условию (8), соответствующую управлению u_k. Будем считать, что пара (x_k, u_k) продолжена на интервале $[T_k, \infty)$ произвольным допустимым образом.

Лемма 1 *Пусть выполнены условия (H1) - (H4), u_* - оптимальное управление в задаче (P), $\{(P_k)\}$, $k = 1, 2, \ldots$ - последовательность задач, соответствующая u_*, и пусть u_k - оптимальное управление в задаче (P_k). Тогда для любого фиксированного $T > 0$ имеем сходимость u_k к u_* в $L^2([0,T], \mathbb{R}^m)$ при $k \to \infty$.*

Доказательство леммы 1 проводится аналогично доказательству леммы 1 в [5] и леммы 7.1 в [1].

Так как пара (x_k, u_k) оптимальна в задаче (P_k), $k = 1, 2, \ldots$ на конечном интервале времени, то для неё выполняется принцип максимума Понтрягина [3]: существует такая пара сопряжённых переменных (ψ_k, ψ_k^0), соответствующая (x_k, u_k), что $\psi_k^0 > 0$ и пара (ψ_k, ψ_k^0) удовлетворяет на $[0, T_k]$ сопряжённой системе

$$\dot{\psi}_k(t) = -\left[\frac{\partial f(x_k(t), t, u_k(t))}{\partial x}\right]^* \psi_k(t) - \psi_k^0 e^{-\rho t} \frac{\partial g(x_k(t), u_k(t))}{\partial x} \tag{9}$$

и условию максимума

$$\mathcal{H}_k(x_k(t), t, u_k(t), \psi_k(t), \psi_k^0) \overset{\text{п.в.}}{=} H_k(x_k(t), t, \psi_k(t), \psi_k^0), \tag{10}$$

где

$$\mathcal{H}_k(x, t, u, \psi, \psi^0) = \langle f(x, t, u), \psi \rangle + \psi^0 e^{-\rho t} g(x, u) - \psi^0 \frac{e^{-t}}{1 + \sigma_k} \|u - z_k(t)\|^2, \tag{11}$$

$$H_k(x, t, \psi, \psi^0) = \max_{u \in U} \mathcal{H}_k(x, t, u, \psi, \psi^0).$$

Кроме того, выполняется условие трансверсальности

$$\psi_k(T_k) = 0. \tag{12}$$

Из соотношений (9) и (10) следует (см. [3]), что на отрезке $[0, T_k]$

$$\frac{d}{dt} H_k(x_k(t), t, \psi_k(t), \psi_k^0) \overset{\text{п.в.}}{=} \frac{\partial \mathcal{H}_k}{\partial t}(x_k(t), t, u_k(t), \psi_k(t), \psi_k^0). \tag{13}$$

Будем считать, что для любого $k = 1, 2, \ldots$ функция ψ_k продолжена на $[0, \infty)$ нулём по непрерывности, т.е. $\psi_k(t) = 0$ при $t \geq T_k$.

Лемма 2 *Пусть выполнены условия (H1) - (H4), (x_*, u_*) - оптимальная пара в задаче (P); $\{(P_k)\}$, $k = 1, 2, \ldots$- последовательность задач, соответствующая u_*, (x_k, u_k) - оптимальная пара в задаче (P_k) при всех $k = 1, 2, \ldots$; (ψ_k, ψ_k^0) - пара сопряжённых переменных, соответствующих (x_k, u_k), так, что выполняются соотношения принципа максимума (9) - (13); последовательности $\{\psi_k(0)\}$ и $\{\psi_k^0\}$ ограничены и выполняется условие*

$$\|\psi_k(0)\| + \psi_k^0 \geq a \quad (k = 1, 2, \ldots) \tag{14}$$

при некотором $a > 0$.

Тогда существует такая подпоследовательность последовательности $\{(x_k, u_k, \psi_k, \psi_k^0)\}$, $k = 1, 2, \ldots$, что для любого фиксированного $T > 0$ выполняются соотношения:

1. $\lim\limits_{k \to \infty} u_k(t) \overset{\text{п.в.}}{=} u_*(t)$ *на отрезке* $[0, T]$;
2. $x_k \rightrightarrows x_*$ *на отрезке* $[0, T]$ *при* $k \to \infty$;
3. $\lim\limits_{k \to \infty} \psi_k^0 = \psi^0$;

4. $\psi_k(t) \rightrightarrows \psi(t)$, *на отрезке* $[0,T]$ *при* $k \to \infty$, *где* (ψ, ψ_0) - *нетривиальная пара сопряжённых переменных, соответствующая* (x_*, u_*);

5. *Пара* (ψ, ψ^0) *удовлетворяет соотношениям*

$$\dot{\psi}(t) = -\left[\frac{\partial f(x_*(t), t, u_*(t))}{\partial x}\right]^* \psi(t) - \psi^0 e^{-\rho t} \frac{\partial g(x_*(t), u_*(t))}{\partial x}; \tag{15}$$

$$\|\psi(0)\| + \psi^0 > 0; \tag{16}$$

$$\mathcal{H}(x_*(t), t, u_*(t), \psi(t), \psi^0) \overset{п.в.}{=} H(x_*(t), t, \psi(t), \psi^0). \tag{17}$$

Доказательство леммы 2 проводится аналогично доказательству леммы 2 из [5] и теоремы 7.1 из [1].

Таким образом, при выполнении условий Леммы 2, можно осуществлять предельный переход в сопряжённой системе и в условии максимума, аппроксимируя исходную задачу (P) на бесконечном интервале времени последовательностью задач $\{(P_k)\}, k = 1, 2, \ldots$ на конечных интервалах.

3 Принцип максимума Понтрягина для неавтономных систем. Случай монотонной зависимости функционала и правой части уравнения от фазовой переменной.

В данном разделе рассматривается случай, в котором правая часть уравнения в задаче (1), а также подынтегральная функция в критерии качества монотонно зависят от переменной x. Т.е. выполняется условие:

Н 5 *Для любой допустимой пары* (x, u) *при почти всех* $t > 0$ *выполняются неравенства:*

$$\frac{\partial g(x(t), u(t))}{\partial x^i} > 0,$$

и

$$\frac{\partial f^i(x(t), t, u(t))}{\partial x^j} \geq 0 \text{ для любых } i, j = 1 \ldots n$$

Покажем, что в этом случае принцип максимума Понтрягина в нормальной форме ($\psi^0 = 1$) является необходимым условием оптимальности для задачи (1). Кроме того, все координаты сопряжённого вектора $\psi(t)$ положительны.

Теорема 1 *(Принцип максимума Понтрягина) Пусть* (x_*, u_*) - *оптимальная пара в задаче оптимального управления на бесконечном интервале времени. Пусть, кроме того, верны условия Н1-Н4, а также условие Н5; интеграл*

$$\int_0^\infty \frac{\partial f(x(t), t, u(t))}{\partial t} dt$$

сходится абсолютно для любой допустимой пары (x, u)*; Существует такое значение* $u_0 \in U$, *что выполняется неравенство*

$$\int_0^\infty \frac{\partial f(x(t), t, u(t))}{\partial t} dt < f(x_0, 0, u_0); \tag{18}$$

Тогда существуют такая абсолютно непрерывная векторная функция $\psi : [0, \infty) \to \mathcal{R}^n$, *что пара* (x_*, u_*) *вместе с* $(\psi, \psi^0 = 1)$ *удовлетворяет условиям:*

1) *функция* ψ *является решением сопряженной системы (15);*

2) *выполняется условие максимума (17);*
3) *выполняется условие стационарности в форме*

$$H(x_*(t),t,\psi(t),1) = -\int_t^\infty \left\langle \frac{\partial f(x_*(s),s,u_*(s))}{\partial s},\psi(s)\right\rangle ds + \rho\int_t^\infty e^{-\rho s}g(x_*(s),u_*(s))ds, \quad (19)$$

4) *сопряжённая переменная $\psi(t)$ удовлетворяет неравенству*

$$\psi(t) > 0, \quad \forall t \geq 0. \quad (20)$$

Доказательство. Рассмотрим последовательность задач $\{(P_k)\}, k = 1,2,\ldots$, аппроксимирующую задачу (P). Пусть пара (x_k,u_k) оптимальна в задаче (P_k). Для задач на конечном интервале времени верно необходимое условие оптимальности в форме принципа максимума Понтрягина [3]. Пусть ψ_k - сопряжённая переменная, соответствующая паре (x_k,u_k) (т.е. ψ_k, x_k, u_k удовлетворяют соотношениям принципа максимума (9)-(13)).

Выполнение неравенства

$$\psi_k(t) > 0, \quad \forall t \in [0,T_k). \quad (21)$$

доказывается аналогично [1].

Теперь покажем, что последовательность $\{\psi_k(0)\}$ ограничена. Для доказательства воспользуемся соотношением

$$\frac{\partial \mathcal{H}}{\partial t}(x_k(t),t,u_k(t),\psi_k(t)) = \left\langle \frac{\partial f(x_k(t),t,u_k(t))}{\partial t},\psi_k(t)\right\rangle - \rho e^{-\rho t}g(x_k(t),u_k(t)) + \quad (22)$$
$$+\frac{e^{-t}}{1+\sigma_k}\|u_k(t)-z_k(t)\|^2 + 2\frac{e^{-t}}{1+\sigma_k}\langle u_k(t)-z_k(t),\dot z_k(t)\rangle.$$

Проинтегрировав последнее равенство от t до T_k, получаем

$$H_k(x_k(t),t,\psi_k(t)) = \max_{u\in U}\left[e^{-\rho T_k}g(x_k(T_k),u) - \frac{e^{-T_k}}{1+\sigma_k}\|u-z_k(T_k)\|^2\right] - \quad (23)$$
$$-\int_t^{T_k}\left\langle \frac{\partial f(x_k(s),s,u_k(s))}{\partial s},\psi_k(s)\right\rangle ds + \rho\int_t^{T_k} e^{-\rho s}g(x_k(s),u_k(s))ds -$$
$$-\frac{1}{1+\sigma_k}\int_t^{T_k} e^{-s}\|u_k(s)-z_k(s)\|^2 ds - 2\frac{1}{1+\sigma_k}\int_t^{T_k} e^{-s}\langle u_k(s)-z_k(s),\dot z_k(s)\rangle ds.$$

.

Из соотношения (23), следует неравенство

$$H_k(x_0,0,\psi_k(0)) \leq M + \int_0^\infty \left\langle \frac{\partial f(x_k(s),s,u_k(s))}{\partial t},\psi_k(s)\right\rangle ds. \quad (24)$$

Из условия монотонности (H5) и сопряжённого уравнения (9), а также из (21) получим, что $\psi_k(t)$ на отрезке $[0,T_k)$ положительна и монотонно убывает. Следовательно,

$$\psi_k(t) \leq \psi_k(0), \quad \forall t \in [0,T_k].$$

Так как $H_k(x_0,0,\psi_k(0))$ - максимальное значение функции Гамильтона-Понтрягина, от неравенства (24) можно перейти к неравенству

$$\langle f(x_0,0,u_0),\psi_k(0)\rangle + g(x_0,u_0) - \frac{\|u_0-z_k(0)\|^2}{1+\sigma_k} \leq M + \langle \psi_k(0), \int_0^\infty \frac{\partial f(x_k(s),s,u_k(s))}{\partial t}ds\rangle$$

отсюда, воспользовавшись условием о сходимости интеграла и неравенством (18), получим ограниченность последовательности $\psi_k(0)$. Так как функции $\psi_k(t)$ монотонно убывают, из

ограниченности последовательности $\{\psi_k(0)\}$ следует что последовательность функций $\{\psi_k(t)\}$ равномерно ограничена.

Таким образом, выполняются условия леммы 2 из которой следует существование векторной функции $\psi(t)$ и выполнение соотношений 1) и 2). Теперь покажем, что выполняется условие стационарности (19).

Для этого воспользуемся следствием (13) из принципа максимума Понтрягина (см. [3]). Рассмотрим отдельно каждое слагаемое в выражении (23).

В силу (4) - (6) третье, четвёртое и пятое слагаемые стремятся к 0 при $k \to \infty$;

Условие

$$\lim_{T_k \to \infty} e^{-\rho T_k} g(x_k(T_k), u) = 0, \quad \forall u \in U$$

выполнено в силу выполнения условия (H4). Следовательно, первое слагаемое тоже стремится к 0 при $k \to \infty$.

В силу условия (H4) для каждого $\varepsilon > 0$ найдётся такое $\overline{T} > 0$, что для любой допустимой пары (x, u) выполняется неравенство

$$\int_T^\infty e^{-\rho s} \left| g(x(s), u(s)) \right| ds < \varepsilon \tag{25}$$

при всех $T \geq \overline{T}$. Кроме того, из условия (H4) следует, что в интеграле

$$\int_t^T e^{-\rho s} g(x_k(s), u_k(s)) ds$$

подынтегральная функция ограничена сверху суммируемой функцией. Следовательно, по теореме Лебега (см. [2]) на конечном интервале времени $[t, \bar{T}]$ можно перейти к пределу под знаком интеграла, воспользовавшись леммой 2 о сходимости и непрерывностью функции g. Т. е. для любого $\varepsilon > 0$ найдётся такой номер $\bar{k} > 0$ что, для всех $k \geq \bar{k}$, выполняется неравенство

$$\left| \int_t^{\bar{T}} e^{-\rho s} g(x_k(s), u_k(s)) ds - \int_t^{\bar{T}} e^{-\rho s} g(x_*(s), u_*(s)) ds \right| < \varepsilon, \tag{26}$$

следовательно,

$$\begin{aligned} &\left| \int_t^{T_k} e^{-\rho s} g(x_k(s), u_k(s)) ds - \int_t^{\infty} e^{-\rho s} g(x_*(s), u_*(s)) ds \right| \leq \\ &\leq \left| \int_t^{\bar{T}} e^{-\rho s} g(x_k(s), u_k(s)) ds - \int_t^{\bar{T}} e^{-\rho s} g(x_*(s), u_*(s)) ds \right| + \\ &+ \left| \int_{\bar{T}}^{T_k} e^{-\rho s} g(x_k(s), u_k(s)) ds - \int_{\bar{T}}^{\infty} e^{-\rho s} g(x_*(s), u_*(s)) ds \right| \leq 4\varepsilon, \end{aligned} \tag{27}$$

начиная с некоторого k в силу выполнения неравенств (25) и (26), а также в силу неравенства

$$\left| \int_{\bar{T}}^{T_k} e^{-\rho s} g(x_k(s), u_k(s)) ds \right| \leq \left| \int_{\bar{T}}^{\infty} e^{-\rho s} g(x_k(s), u_k(s)) ds \right| + \left| \int_{T_k}^{\infty} e^{-\rho s} g(x_k(s), u_k(s)) ds \right| \leq 2\varepsilon$$

при $T_k \geq \bar{T}$.

Таким образом, доказано что

$$\lim_{k \to \infty} \rho \int_t^{T_k} e^{-\rho s} g(x_k(s), u_k(s)) ds = \rho \int_t^{\infty} e^{-\rho s} g(x_*(s), u_*(s)) ds, \tag{28}$$

Рассмотрим слагаемое

$$\int_t^{T_k} \left\langle \frac{\partial f(x_k(s), s, u_k(s))}{\partial s}, \psi_k(s) \right\rangle ds. \tag{29}$$

Докажем, что в выражении (29) также можно перейти к пределу при k стремящемся к бесконечности. Из равномерной ограниченности $\{\psi_k(t)\}$ и сходимости интеграла $\int_0^\infty \frac{\partial f(x(s),s,u(s))}{\partial s} ds$ следует, что для каждого $\varepsilon > 0$ найдётся такое $\overline{T} > 0$, что для любой допустимой пары (x,u) выполняется неравенство

$$\left| \int_T^\infty \left\langle \frac{\partial f(x(s),s,u(s))}{\partial s}, \psi(s) \right\rangle ds \right| < C \left| \int_T^\infty \frac{\partial f(x(s),s,u(s))}{\partial s} ds \right| < \varepsilon$$

для всех $T \geq \overline{T}$. Из этих же условий получим, что подынтегральная функция ограничена сверху суммируемой функцией, поэтому на конечном интервале $[t,\bar{T}]$ применима теорема Лебега (см. [2]).

В результате, получим что

$$\left| \int_t^{T_k} \left\langle \frac{\partial f(x_k(s),s,u_k(s))}{\partial s}, \psi_k(s) \right\rangle ds - \int_t^\infty \left\langle \frac{\partial f(x_*(s),s,u_*(s))}{\partial s}, \psi_*(s) \right\rangle ds \right| \leq$$

$$\left| \int_t^{\bar{T}} \left\langle \frac{\partial f(x_k(s),s,u_k(s))}{\partial s}, \psi_k(s) \right\rangle ds - \int_t^{\bar{T}} \left\langle \frac{\partial f(x_*(s),s,u_*(s))}{\partial s}, \psi_*(s) \right\rangle ds \right| +$$

$$+ \left| \int_{\bar{T}}^{T_k} \left\langle \frac{\partial f(x_k(s),s,u_k(s))}{\partial s}, \psi_k(s) \right\rangle ds - \int_{\bar{T}}^\infty \left\langle \frac{\partial f(x_*(s),s,u_*(s))}{\partial s}, \psi_*(s) \right\rangle ds \right| \leq 4\varepsilon,$$

начиная с некоторого k. Следовательно,

$$\lim_{k\to\infty} \int_t^{T_k} \left\langle \frac{\partial f(x_k(s),s,u_k(s))}{\partial s}, \psi_k(s) \right\rangle ds = \int_t^\infty \left\langle \frac{\partial f(x_*(s),s,u_*(s))}{\partial s}, \psi_*(s) \right\rangle ds.$$

В результате предельного перехода при $k \to \infty$, получим искомое условие стационарности, определяющее поведение гамильтониана на бесконечности (19). Теорема доказана.

Список литературы

[1] С. М. Асеев, А. В. Кряжимский. *Принцип максимума Понтрягина и задачи оптимального экономического роста.* М.: Наука - МАИК Наука. Интерпериодика, 2007. - 272 с. - (Труды МИАН; Т. 257)

[2] А. Н. Колмогоров, С. В. Фомин. *Элементы теории функций и функционального анализа.* Москва, Наука, 1976.

[3] Л. С. Понтрягин, В. Г. Болтянский, Р. В. Гамкрелидзе, Е. Ф. Мищенко. *Математическая теория оптимальных процессов.* Наука. 1979.

[4] А. Ф. Филиппов. *Некоторые вопросы теории оптимального управления.* Вестник Московского университета сер. математика, механика, физика, астрономия 2, 1959.

[5] S. M. Aseev, A. V. Kryazhimskiy. *The Pontryagin Maximum Principle and transversality conditions for a class of optimal control problems with infinite time horizons.* SIAM J. Optimal Control. Vol.43, No 3, 2004, pp.1094-1119

[6] E. J. Balder, *A existence result for optimal economic growth problems*, J. Math. Anal. Appl.,95(1983) pp. 195-213.

[7] D. A. Carlson, A. B. Haurie, and A. Leizarowitz, *Infinite horizon optimal control. Determenistic and Stochastic Systems.* Springer-Verlag, Berlin, 1991.

[8] D. Combs, M. A. Gilchrist, J. Percus, A. S. Perelson, *Optimal Viral Production*, Bulletin of Mathematical Biology, 2003, Vol. 65, No 6, pp. 1003-1023.

[9] H. Halkin, *Necessary conditions for optimal control problems with infinite horizons*, Econometrica, Vol. 42, 1974, pp. 267–272.

[10] P. Michel *On the transversality conditions in infinite horizon optimal problems. Econometrica*, 50, 1982.

УДК 517.927.25

О ВЛИЯНИИ СТЕПЕНИ СУММИРУЕМОСТИ КОЭФФИЦИЕНТОВ ДИФФЕРЕНЦИАЛЬНОГО ОПЕРАТОРА НА СКОРОСТЬ РАВНОСХОДИМОСТИ СПЕКТРАЛЬНЫХ РАЗЛОЖЕНИЙ.

© 2008 г. А. С. Марков

markov.cs@gmail.com

Кафедра Общей математики

1 Введение

Рассмотрим произвольный дифференциальный оператор L, порожденный дифференциальной операцией

$$lu \equiv u'' + p_1(x)u' + q_1(x)u, \qquad x \in G = (0,1),$$

на классе функций D, абсолютно непрерывных на $\overline{G} = [0,1]$ вместе со своей первой производной;

$$p_1(x) \in \mathcal{L}^s(G,\mathcal{C}), \qquad s \geqslant 1, \qquad q_1(x) \in \mathcal{L}(G,\mathcal{C}). \tag{1}$$

Ранее были установлены оценки скорости сходимости и равносходимости с тригонометрическим рядом Фурье спектральных разложений широкого класса функций по корневым функциям некоторого сужения оператора L в метрике пространств $\mathcal{L}^p, p \in [1,\infty]$, на всем интервале G (в работе [1]) и на любом внутреннем отрезке (в работе [2]). Сужение оператора определяется тремя известными *условиями Ильина*. Предполагалось, что нормированные коэффициенты Фурье f_k имеют асимптотику $O(\lambda_k^{-\nu}), \nu = const > 0, k \to \infty, \lambda_k -$ спектральный параметр. В настоящей работе получены аналогичные оценки в случае, когда f_k имеют асимптотику $O(\lambda_k^{-\nu}\ln^{-\beta}|\lambda_k|), |\lambda_k| > 1$. Показано, что число $s-$ степень суммируемости функции $p_1(x)-$ может влиять на скорость равносходимости.

2 Постановка задачи

Дадим определение корневых функций (т.е. собственных и присоединенных функций) и приведем основные условия на операторы. Корневые функции оператора L определим в обобщенном (по Ильину) смысле, рассматривая их как регулярные решения дифференциальных уравнений и освобождая их от требования удовлетворения каким-либо конкретным краевым условиям. Ограничения при этом налагаются на свойства спектра и корневых функций оператора. Это позволяет изучать как системы функций типа системы экспонент, не удовлетворяющие никаким краевым условиям без спектрального параметра, так и корневые функции конкретных краевых задач. Рассмотрены два встречающихся типа спектральных задач (отличающиеся нормировкой присоединенных функций). Под *собственной функцией оператора* L, отвечающей собственному значению $\lambda^2 \in \mathcal{C}$, будем понимать любую не равную тождественному нулю функцию $\overset{0}{u}(x) \in D$, удовлетворяющую почти всюду в G уравнению $l\overset{0}{u} + \lambda^2\overset{0}{u} = 0$. Под *присоединенной функцией порядка m, m=1,2,...*, отвечающей тому же λ^2 и собственной функции $\overset{0}{u}$, будем понимать любую функцию $\overset{m}{u}(x)$, которая почти всюду в G удовлетворяет уравнению $l\overset{m}{u} + \lambda^2\overset{m}{u} = \mu_m\overset{m-1}{u}$. Здесь либо $\mu_m = 1$ (спектральная задача 1), либо $\mu_m = \lambda(\mathbf{Re}\lambda \geqslant 0)$ при $|\lambda| \geqslant 1$, $\mu = 1$ при $|\lambda| < 1$ (спектральная задача 2). Приведем основные ограничения на рассматриваемые системы корневых функций. В случае конкретных краевых задач эти условия достаточно легко проверяются.

Фиксируем произвольную систему собственных значений $\{\lambda_k^2\}_{k=1}^{\infty}$ и произвольную систему $u_k(x)$ корневых функций оператора L, отвечающую этим собственным значениям, удовлетворяющие следующим трем *условиям Ильина,* назовем их *условиями A:*

1) система $\{u_k(x)\}$ замкнута и минимальна в $\mathcal{L}^r(G)$ при некотором $r \in [1, \infty]$;

2) существуют $c_1, c_2 = const > 0$ такие, что

$$|\mathbf{Im}\lambda_k| \leqslant c_1 \quad \forall k; \qquad \sum_{0 \leqslant |\lambda_k| - \lambda \leqslant 1} 1 \leqslant c_2 \quad \forall \lambda \geqslant 0; \tag{2}$$

3) существует $c_3 = const > 0$ такая, что

$$||u_k||_r ||v_k||_{r'} \leqslant c_3 \quad \forall k, \tag{3}$$

где $\{v_k\}-$ биортогонально сопряженная с $\{u_k\}$ система функций: $v_k \in \mathcal{L}^{r'}(G)$, $(u_k, v_j) = \delta_{kj} \quad \forall k, j \in \mathcal{N}$, $r' = r/(r-1)$; $||\cdot||_r-$ обозначение нормы в $\mathcal{L}^r(G)$.

Будем также обозначать через $||f||_{r,E}$ норму функции $f(x) \in \mathcal{L}^r(E)$, $r \in [1, \infty]$.

Для произвольной функции $f(x) \in \mathcal{L}^r(G)$ составим частичные суммы биортогонального разложения

$$\sigma_\lambda(x, f) = \sum_{|\lambda_k| \leqslant \lambda} f_k u_k(x), \quad \lambda > 0, \quad f_k \equiv (f, v_k).$$

Через $S_\lambda(x, f)$ обозначим частичную сумму тригонометрического ряда Фурье функции $f(x)$, рассматриваемого как ортогональное разложение $f(x)$ для оператора $L_0 u = u''$ с условиями периодичности в 0 и 1.

Наложим дополнительно ограничение на систему $\{u_k, v_k\}$ и функции $f(x)$:

$$\exists \nu, \beta = const > 0 : \qquad \alpha_k f_k = O(\lambda_k^{-\nu} \ln^{-\beta} |\lambda_k|), \quad |\lambda_k| > 1, \tag{4}$$

где $\alpha_k = ||v_k||_{r'}^{-1}$. Предположим далее, что для оператора L_0 условие (4) выполняется.

Основная задача состоит в том, чтобы установить факт равносходимости спектральных разложений $\sigma_\lambda(x, f)$ и $S_\lambda(x, f)$ функции $f(x)$ в метрике пространств $\mathcal{L}^p$, $p \geqslant 1$, на всём интервале и на любом внутреннем отрезке $K \subset G$ и оценить скорость равносходимости указанных разложений (или, другими словами, оценить погрешность аппроксимации одного разложения другим). Из этой оценки будет следовать, в частности, что оба разложения сходятся или расходятся в метрике данного пространства $\mathcal{L}^p$ одновременно. Требуется при этом выявить степень зависимости оценок скорости равносходимости от показателя s.

3 Равносходимость на внутреннем компакте

Теорема 1. *Пусть для оператора L и функции $f(x)$ выполняются условия (1),(4) и условия A, $s \neq 1$. Тогда для всех достаточно больших чисел λ и любого отрезка $K \subset G$ справедлива оценка*

$$||\sigma_\lambda(x, f) - S_\lambda(x, f)||_{\mathcal{L}^p(K)} \leqslant c \max(\lambda^{-1}, \lambda^{-\nu} \ln^{-\beta+2} \lambda, \lambda^{-\nu-1/p+1/s} \ln^{-\beta+1} \lambda, \lambda^{-\nu-1/p+1/s}) \tag{5}$$

с постоянной c, не зависящей от λ.

Пусть $\Theta_\lambda(x, y) = \sum_{|\lambda_k| \leqslant \lambda} u_k(x)\overline{v}_k(y)-$ спектральная функция оператора L, $D_\lambda(x, y)-$ ядро Дирихле, спектральная функция оператора L_0; $\sigma_\lambda(x, f) = \int_{\overline{G}} \Theta_\lambda(x, y) f(y) dy$. Фиксируем любой отрезок $K = [a, b] \subset G$. Нас интересует оценка разности при $\lambda \to +\infty$:

$$\Delta_\lambda \equiv \int_K |\sigma_\lambda(x, f) - S_\lambda(x, f)|^p dx = \int_K \left| \int_{\overline{G}} f(y)[\Theta_\lambda(x, y) - D_\lambda(x, y)] dy \right|^p dx. \tag{6}$$

Зафиксировав произвольное число $R_0 \in (0, (1/2)\mathrm{dist}(K, \partial G))$, для любого числа $R \in [R_0/2, R_0]$ и любого $x \in K$ рассмотрим вспомогательную функцию

$$\omega_\lambda(\rho, R) = \begin{cases} (\sin\lambda\rho)/(\pi\rho), & |\rho| \leqslant R, \rho = x - y, \\ 0, & |\rho| > R. \end{cases} \tag{7}$$

Далее, как и в работах В.А. Ильина [3,4], усредним функцию (7) по R :

$$S_0[\omega_\lambda(\rho, R)] = \frac{8}{3R_0^2} \int\limits_{R_0/2}^{R_0} R\omega_\lambda(\rho, R)dR \quad (S_0[1] = 1), \tag{8}$$

добавим и вычтем под знаком модуля в правой части (6) выражение (8) и оценим модуль суммы через сумму модулей ($c_p = 2^{p-1}$, $v_0(x, y) = S_0[\omega_\lambda(\rho, R)]$) :

$$\Delta_\lambda \leqslant c_p \int\limits_K \left| \int\limits_{\overline{G}} f(y)[\Theta_\lambda(x, y) - v_0(x, y)]dy \right|^p dx + c_p \int\limits_K \left| \int\limits_{\overline{G}} f(y)[D_\lambda(x, y) - v_0(x, y)]dy \right|^p dx \equiv$$

$$\equiv c_p(I + \widehat{I}). \tag{9}$$

Оценим первый интеграл I в правой части (9); интеграл $\widehat{I}$ оценивается аналогично.

В работе [1] показано, что искомая оценка складывается из четырех составляющих: K_1, $||S_1||_{p,K}$, $|S_2(x)|$ и $||S_3||_{p,K}$.

Слагаемое $K_1 = \sum\limits_k |f_k u_k(x) I_k^\lambda(R_0)|$ оценивается следующей суммой:

$$K_1 \leqslant c\left[\lambda^{-2}\sum\nolimits_1 |\hat{f}_k| + \lambda^{-2}\sum\nolimits_2 |\hat{f}_k| + \lambda^{-1}\sum\nolimits_3 |\hat{f}_k\lambda_k^{-1}| + \sum\nolimits_4 |\hat{f}_k| + \sum\nolimits_5 |\hat{f}_k||\lambda - |\lambda_k||^{-2}\right], \tag{10}$$

$$\hat{f}_k \equiv \alpha_k f_k,$$

где суммы $\sum_1$, $\sum_2$, $\sum_3$, $\sum_4$ и $\sum_5$ соответствуют разбиению множества $\{\lambda_k\}$: $\sum_1$ – сумма по тем λ_k, для которых справедливо неравенство $|\lambda_k| < 1$; $\sum_2 : \ 1 < |\lambda_k| \leqslant \lambda/2$; $\sum_3 : \ |\lambda_k| \geqslant 3\lambda/2$; $\sum_4 : \ |\lambda - |\lambda_k|| \leqslant 2/R_0$; $\sum_5 : \ 2/R_0 \leqslant |\lambda - |\lambda_k|| \leqslant \lambda/2$.

Оценим суммы в правой части неравенства (10). Для суммы $\sum_1$ получим оценку $O(\lambda^{-2})$. Сумму $\sum_2$ оценим следующим образом: $\lambda^{-2}\sum_2 |\hat{f}_k| \leqslant c\lambda^{-2}\sum\limits_{n=2}^{[\lambda/2]} n^{-\nu}\ln^{-\beta} n = \varphi(\lambda)$, где $\varphi(\lambda) = O(\lambda^{-2})$ при $\nu > 1$, $\varphi(\lambda) = O(\lambda^{-2}\ln^{-\beta+1}\lambda)$ при $\nu = 1$ и $\varphi(\lambda) = O(\lambda^{-\nu-1})$ при $\nu < 1$. Для суммы $\sum_3$ получаем $\lambda^{-1}\sum_3 |\hat{f}_k\lambda_k^{-1}| \leqslant c\lambda^{-1}\sum\limits_{n \geqslant [3\lambda/2]} n^{-\nu-1}\ln^{-\beta} n = O(\lambda^{-\nu-1}\ln^{-\beta}\lambda)$. Сумма $\sum_4$ имеет, очевидно, оценку $O(\lambda^{-\nu}\ln^{-\beta}\lambda)$, а сумма $\sum_5 |\hat{f}_k||\lambda - |\lambda_k||^{-2} \leqslant O(\lambda^{-\nu}\ln^{-\beta}\lambda)$. Объединяя установленные оценки, получаем

$$\sum_k |f_k u_k(x) I_k^\lambda(R_0)| = O(\max(\lambda^{-2}, \lambda^{-\nu}\ln^{-\beta}\lambda)). \tag{11}$$

Для $||S_1||_{p,K}$ имеем следующее неравенство:

$$||S_1||_{p,K} \leqslant c_0(\lambda^{-1}\sum\nolimits_1 |\hat{f}_k| + \lambda^{-1}\sum\nolimits_2 |\hat{f}_k\lambda_k^{-1/p}| + \lambda^{1/q}\sum\nolimits_3 |\hat{f}_k\lambda_k^{-2}| + \lambda^{-1/p}\sum\nolimits_4 |\hat{f}_k| +$$

$$+\lambda^{1/q}\sum\nolimits_5 |\hat{f}_k\lambda_k^{-1}||\lambda - |\lambda_k||^{-1}), \tag{12}$$

где $p^{-1} + q^{-1} = 1$.

Установим общую оценку для правой части (12). Как и выше, получаем $\lambda^{-1}\sum_1|\hat{f}_k| = O(\lambda^{-1})$. Далее имеем

$$\lambda^{-1}\sum\nolimits_2|\hat{f}_k\lambda_k^{-1/p}| \leqslant c\lambda^{-1}\sum_{n=2}^{[\lambda/2]} n^{-\nu-1/p}\ln^{-\beta} n = \varphi(\lambda),$$

где $\varphi(\lambda) = O(\lambda^{-1})$ при $\nu > 1/q$, $\varphi(\lambda) = O(\lambda^{-1}\ln^{-\beta+1}\lambda)$ при $\nu = 1/q$ и $\varphi(\lambda) = O(\lambda^{-\nu-1/p})$ при $\nu < 1/q$. Для суммы $\sum_3$ находим, что

$$\lambda^{1/q}\sum\nolimits_3|\hat{f}_k\lambda_k^{-2}| \leqslant c\lambda^{1/q}\sum_{n\geqslant 3\lambda/2} n^{-\nu-2}\ln^{-\beta} n = O(\lambda^{-\nu-1/p}\ln^{-\beta}\lambda).$$

Оценка для суммы $\sum_4$ имеет вид $\lambda^{-1/q}\sum_4|\hat{f}_k| = O(\lambda^{-\nu-1/p}\ln^{-\beta}\lambda)$. Наконец, оцениваем сумму $\sum_5$:

$$\lambda^{1/q}\sum\nolimits_5|\hat{f}_k\lambda_k^{-1}||\lambda-|\lambda_k||^{-1} \leqslant c\lambda^{-\nu+1/q-1}\ln^{-\beta}\lambda\sum_5|\lambda-|\lambda_k||^{-1} = O(\lambda^{-\nu-1/p}\ln^{-\beta+1}\lambda).$$

Объединяя установленные оценки, получаем

$$||S_1||_{p,K} = O(\max(\lambda^{-1}, \lambda^{-\nu-1/p}\ln^{-\beta+1}\lambda, \lambda^{-\nu-1/p})). \tag{13}$$

Для суммы $|S_2(x)|$ имеем:

$$|S_2(x)| \leqslant \lambda^{-1}\sum\nolimits_1|\hat{f}_k| + \lambda^{-1}\sum\nolimits_2|\hat{f}_k|\ln|\lambda_k| + \ln\lambda\sum\nolimits_3|\hat{f}_k\lambda_k^{-1}| + \lambda^{-1}\ln\lambda\sum\nolimits_4|\hat{f}_k\lambda_k| +$$
$$+\lambda^{-1}\ln\lambda\sum\nolimits_5|\hat{f}_k\lambda_k||\lambda-|\lambda_k||^{-1}. \tag{14}$$

Оценим правую часть последнего неравенства. $\lambda^{-1}\sum_1|\hat{f}_k| = O(\lambda^{-1})$, далее выводим оценки

$$\lambda^{-1}\sum\nolimits_2|\hat{f}_k|\ln|\lambda_k| \leqslant \lambda^{-1}\sum_{n=2}^{[\lambda/2]} n^{-\nu}\ln^{-\beta+1} n = \varphi(\lambda),$$

где $\varphi(\lambda) = O(\lambda^{-1})$ при $\nu > 1$, $\varphi(\lambda) = O(\lambda^{-1}\ln^{-\beta+2}\lambda)$ при $\nu = 1$ и $\varphi(\lambda) = O(\lambda^{-\nu}\ln\lambda)$ при $\nu < 1$; $\ln\lambda\sum_3|\hat{f}_k\lambda_k^{-1}| = O(\lambda^{-\nu}\ln^{-\beta+1}\lambda)$; $\lambda^{-1}\ln\lambda\sum_4|\hat{f}_k\lambda_k| = O(\lambda^{-\nu}\ln^{-\beta+1}\lambda)$; $\lambda^{-1}\ln\lambda\sum_5|\hat{f}_k\lambda_k||\lambda-|\lambda_k||^{-1} \leqslant \lambda^{-\nu}\ln^{-\beta+1}\sum_5|\lambda-|\lambda_k||^{-1} = O(\lambda^{-\nu}\ln^{-\beta+2}\lambda)$. В результате имеем, что

$$S_2(x) = O(\max(\lambda^{-1}, \lambda^{-\nu}\ln^{-\beta+2}\lambda, \lambda^{-\nu}\ln\lambda)). \tag{15}$$

Для $||S_3||_{p,K}$ имеем следующее неравенство:

$$||S_3||_{p,K} \leqslant c_0||p_1||_s(\lambda^{-1}\sum\nolimits_1|\hat{f}_k| + \lambda^{-1}\sum\nolimits_2|\hat{f}_k\lambda_k^{1-1/\varkappa}| + \lambda^{1-1/\varkappa}\sum\nolimits_3|\hat{f}_k\lambda_k^{-1}| + \lambda^{-1/\varkappa}\sum\nolimits_4|\hat{f}_k\lambda_k| +$$
$$+\lambda^{1-1/\varkappa}\sum\nolimits_5|\hat{f}_k||\lambda-|\lambda_k||^{-1}), \tag{16}$$

где $1/\varkappa = 1 + 1/p - 1/s$. Оценим правую часть последнего неравенства по прежней схеме: $\lambda^{-1}\sum_1|\hat{f}_k| = O(\lambda^{-1})$; $\lambda^{-1}\sum_2|\hat{f}_k\lambda_k^{1-1/\varkappa}| = \varphi(\lambda)$, где $\varphi(\lambda) = O(\lambda^{-1})$ при $\nu > 2 - 1/\varkappa$, $\varphi(\lambda) = O(\lambda^{-1}\ln^{-\beta+1}\lambda)$ при $\nu = 2 - 1/\varkappa$ и $\varphi(\lambda) = O(\lambda^{-\nu-1/\varkappa+1}) = O(\lambda^{-\nu-1/p+1/s})$ при $\nu < 2 - 1/\varkappa$; $\lambda^{1-1/\varkappa}\sum_3|\hat{f}_k\lambda_k^{-1}| = O(\lambda^{-\nu-1/\varkappa+1}\ln^{-\beta}\lambda) = O(\lambda^{-\nu-1/p+1/s}\ln^{-\beta}\lambda)$; такая же оценка имеет место для суммы $\sum_4$;
$\lambda^{1-1/\varkappa}\sum_5|\hat{f}_k||\lambda-|\lambda_k||^{-1} = O(\lambda^{-\nu-1/\varkappa+1}\ln^{-\beta+1}\lambda) = O(\lambda^{-\nu-1/p+1/s}\ln^{-\beta+1}\lambda)$. Таким образом,

$$||S_3||_{p,K} = O(\max(\lambda^{-1}, \lambda^{-\nu-1/p+1/s}\ln^{-\beta+1}\lambda, \lambda^{-\nu-1/p+1/s})). \tag{17}$$

Объединяя оценки (11), (13), (15) и (17), получаем оценку (5) теоремы.

4 Равносходимость на всем отрезке [0,1]

Имеет место следующее утверждение.

Теорема 2. *Пусть для оператора L и функции $f(x)$ выполняются условия (1),(4) и условия A, $s \neq 1$. Тогда для всех достаточно больших чисел λ справедлива оценка*

$$||\sigma_\lambda(x,f) - S_\lambda(x,f)||_{\mathcal{L}^p(G)} \leqslant c\max(\lambda^{-1}, \lambda^{-\nu-1/p+1}, \lambda^{-\nu}\ln^{-\beta+2}\lambda, \lambda^{-\nu-1/p+1/s}\ln^{-\beta+1}\lambda) \qquad (18)$$

с постоянной c, не зависящей от λ.

Введем следующие обозначения: $\Theta_\lambda(x,y) = \sum\limits_{|\lambda_k|\leqslant\lambda} u_k(x)\overline{v}_k(y)-$ спектральная функция оператора L, $D_\lambda(x,y)-$ ядро Дирихле, спектральная функция оператора L_0; $\sigma_\lambda(x,f) = \int\limits_{\overline{G}} \Theta_\lambda(x,y)f(y)dy$. Фиксируем любой отрезок $K_1 = [a,b] \subset G$. Справедливо представление

$$||\sigma_\lambda(x,f) - S_\lambda(x,f)||_p^p = ||\cdot||_{p,(0,a)}^p + ||\cdot||_{p,K_1}^p + ||\cdot||_{p,(b,1)}^p \equiv \Delta_\lambda^a + \Delta_\lambda + \Delta_\lambda^b. \qquad (19)$$

Оценим слагаемое Δ_λ^a (Δ_λ^b оценивается аналогично). Положим $t = (b-a)/4$, $x = z - t, x \in [0,a]$. Тогда $z \in K = [t, a+t] \subset G$. В интеграле Δ_λ^a перейдем к переменной z :

$$\Delta_\lambda^a = \int\limits_K |\sigma_\lambda(z-t,f) - S_\lambda(z-t,f)|^p dz = \int\limits_K \left| \int\limits_G f(y)[\Theta_\lambda(z-t,y) - D_\lambda(z-t,y)]dy \right|^p dz.$$

Далее следует повторить рассуждения (22)-(34) из [1], в результате чего получаем, что итоговая оценка скорости равносходимости разложений на интервале складывается из оценки скорости равносходимости на компакте (теорема 1) и оценок слагаемых Δ_λ^a и Δ_λ^b (формула (29) из [1]). Необходимо оценить следующее выражение, с учетом условий (1),(4) и условия A:

$$\left[\lambda^{-q}\sum\nolimits_1 |\hat{f}_k|^q + \lambda^{-q}\sum\nolimits_2 |\hat{f}_k\lambda_k|^q + \lambda^q\sum\nolimits_3 |\hat{f}_k\lambda_k^{-1}|^q + \ln^q\lambda\sum\nolimits_4 |\hat{f}_k{}^q| + \sum\nolimits_5 |\hat{f}_k|^q \ln^q \frac{\lambda}{|\lambda - |\lambda_k||}\right]^{p/q}, \qquad (20)$$

Для суммы $\sum_1$ получим оценку $O(\lambda^{-1})$. Сумму $\sum_2$ оценим следующим образом: $\lambda^{-q}\sum_2 |\hat{f}_k\lambda_k|^q \leqslant c\lambda^{-q}\sum\limits_{n=2}^{[\lambda/2]} n^{(-\nu+1)q}\ln^{-\beta q} n = \varphi(\lambda)$, где $\varphi(\lambda) = O(\lambda^{-q})$ при $\nu > 1 + 1/q$, $\varphi(\lambda) = O(\lambda^{-q}\ln^{-\beta q+1}\lambda)$ при $\nu = 1 + 1/q$ и $\varphi(\lambda) = O(\lambda^{-\nu q+1})$ при $\nu < 1 + 1/q$. Для суммы $\sum_3$ получаем $\lambda^{-q}\sum_3 |\hat{f}_k\lambda_k^{-1}|^q \leqslant O(\lambda^{-\nu q+1}\ln^{-\beta q}\lambda)$. Сумма $\sum_4$ имеет, очевидно, оценку $O(\lambda^{-\nu q}\ln^{-\beta q+q}\lambda)$, а сумма $\sum_5 |\hat{f}_k|^q \ln^q \frac{\lambda}{|\lambda-|\lambda_k||} \leqslant O(\lambda^{-\nu q+1})$. Объединяем полученные оценки, затем подставляем их в (20), и далее, извлекая корень степени p, получаем оценки слагаемых Δ_λ^a и Δ_λ^b. Учитывая результаты теоремы 1 (оценка слагаемого Δ_λ), получаем оценку (18).

Автор признателен И.С. Ломову за постановку задачи и полезные замечания в процессе выполнения работы.

Список литературы

[1] Ломов И.С. - *Дифференц. уравнения* 1996. Т. 32. №1. С. 71-82

[2] Ломов И.С. - *Дифференц. уравнения* 1998. Т. 34. №5. С. 619-628

[3] Ильин В.А. - *Дифференц. уравнения* 1980. Т. 16. №5. С. 771-194; №6. С. 980-1006

[4] Ильин В.А. - *Доклад РАН* 1983. Т. 273. №4. С. 789-793

УДК 004.056.5

СТЕГАНОГРАФИЧЕСКИЙ АЛГОРИТМ ВНЕДРЕНИЯ ИНФОРМАЦИИ В ФАЙЛЫ ФОРМАТА BMP, УСТОЙЧИВЫЙ К СЖАТИЮ JPEG

© 2008 г. А.А.Мартыненко

amartynenko@list.ru

Кафедра Математической Кибернетики

1 Введение.

Стеганография — это наука о скрытой передаче информации путём сохранения в тайне самого факта передачи информации. В настоящее время под стеганографией чаще всего понимают скрытие информации в текстовых, графических, аудио и видео файлах путём использования специального программного обеспечения.

Цифровая стеганография [1] как наука родилась буквально в последние годы. Она включает в себя следующие направления:
1) встраивание информации с целью ее скрытой передачи;
2) встраивание цифровых водяных знаков (ЦВЗ) (watermarking);
3) встраивание идентификационных номеров (fingerprinting);
4) встраивание заголовков (captioning).

В связи с бурным развитием информационных технологий остро встал вопрос о защите авторских прав для мультимедийных продуктов: изображений, аудио и видеозаписей. Одной из самых эффективных технологий защиты является встраивание ЦВЗ — невидимых меток, представляющих собой некоторый фиксированный набор байт. Технология встраивания идентификационных номеров очень похожа на технологию ЦВЗ. Отличием является то, что каждая копия мультимедийного файла содержит свой определенный ЦВЗ, который каким-либо образом связан с тем лицом, которому выдается данная копия. Технология встраивания заголовков применяется тогда, когда нужно хранить информацию разного рода в одном контейнере. Например, хранить диагноз пациента в его же рентгеновском снимке.

В противодействие этим технологиям нарушитель может использовать достаточно большое количество методов для уничтожения информации, встроенной в статическое изображение. К этим методам относятся изменение разрешения изображения, добавление различных фильтров, шумов, изменение цвета в некоторых областях изображения, сжатие изображения и т.д. В данной статье предложен стеганографический алгоритм внедрения ЦВЗ в изображение формата BMP устойчивый к JPEG сжатию.

2 Формат BMP файла и сжатие JPEG.

В начале файла формата BMP идут два заголовка: FILE HEADER и INFORMATION HEADER. В них содержится информация о размере файла, разрешении изображения, количестве битов на пиксель и так далее. Будем рассматривать только такие изображения формата BMP, для которых количество битов на пиксель равно 24. Это значит, что любой пиксель задается при помощи трех байт, отвечающих за красную (Red), зеленую (Green) и синюю (Blue) составляющие цвета пикселя. Такое задание цвета получило название RGB.

JPEG [2] — один из самых новых и достаточно мощных алгоритмов. Практически он является стандартом де-факто для полноцветных изображений. Оперирует алгоритм областями 8x8, на которых яркость и цвет меняются сравнительно плавно. Алгоритм разработан группой экспертов в области фотографии специально для сжатия 24-битных изображений. JPEG — Joint Photographic Expert Group — подразделение в рамках ISO — Международной организации по стандартизации. В целом алгоритм основан на дискретном косинусоидальном

преобразовании, применяемом к матрице изображения для получения некоторой новой матрицы коэффициентов. JPEG представляет собой сжатие информации с потерями. В начале алгоритма сжатия мы переводим изображение из цветового пространства RGB в цветовое пространство YCrCb (иногда называют YUV). В нем Y — яркостная составляющая, а Cr, Cb — компоненты, отвечающие за цвет (хроматический красный и хроматический синий). За счет того, что человеческий глаз менее чувствителен к цвету, чем к яркости, появляется возможность архивировать массивы для Cr и Cb компонент с большими потерями и, соответственно, большими коэффициентами сжатия, не сжимая при этом яркостную компоненту Y.

3 Алгоритм устойчивый к сжатию JPEG.

Постановка задачи: разработать алгоритм, включающий служебную информацию в файл формата BMP, сжать этот мультимедийный файл в JPEG, "разжать" обратно в BMP, алгоритм после данных преобразований должен быть способен извлечь служебную информацию.

Данный алгоритм был реализован для работы только с файлами формата BMP, так как работать с ними на порядок проще чем с файлами формата JPEG. В ходе работы данного алгоритма, сразу же после JPEG сжатия можно извлечь служебную информацию, не переводя изображение в формат BMP, но для этого, очевидно, нужно уметь работать с файлами JPEG.

Одним из самых популярных стеганографических методов внедрения информации является метод Less Significant Bit(LSB), который вносит изменения в наименее значащие биты, определяющие пиксель. Изменения вносятся в младшие биты, отвечающие красной, зеленой и синей компонентам цвета. Преимущество данного метода в простоте реализации и незаметности изменений изображения для глаза человека, даже при условии того, что человек может одновременно наблюдать и оригинал изображения и изображение с уже внедренным в него ЦВЗ.

Экспериментально проверено, что метод LSB не подходит для алгоритма, устойчивого к сжатию JPEG. Это так, потому что JPEG сжатие уничтожает значения тех бит, которые не вносят существенных изменений в восприятие данного изображения, таким образом уничтожают те самые наименее значащие биты, в которые метод LSB предлагает вносить изменения. Если рассмотреть квадрат из 8x8 пикселей (некоторую квадратную область изображения), итого $8 * 8 * 3 = 192$ байта и в каждый наименее значащий бит (8-й бит) этих байт записать, например, "0", то число ошибок вследствие трансформации BMP – JPEG – BMP может быть как больше 96, так и меньше 96. Примерно такое же количество ошибок возникает если записывать служебную информацию и в 7-й, и в 6-й биты. Только при записи в 5-й бит, JPEG сжатие уничтожает менее половины значений битов. Проблема в том, что при записи в 5-й бит изображение существенно, а главное неестественно искажается, что становится заметно даже невооруженным глазом. Замечательным свойством JPEG сжатия является то, что оно не меняет яркости блоков пикселей размером 8x8.

Пусть $R(P)$, $G(P)$, $B(P)$ — значение байт отвечающих за красный, зеленый и синий цвет пикселя P соответственно. Яркость пикселя $L(P)$ вычисляется по следующей формуле:

$$L(P) = 0.299 * R(P) + 0.587 * G(P) + 0.114 * B(P).$$

Можно заметить, что сумма коэффицентов в этой формуле равна 1. Данный факт используется в дальнейшем.

Яркость блока пикселей размера 8x8 вычисляется как средняя яркость пикселей этого блока $L(BLOCK) = \frac{\sum_{i=1}^{8}\sum_{j=1}^{8} L(P_{i,j})}{64}$. Именно величину $L(BLOCK)$ JPEG сжатие и оставляет нетронутым. Очевидно, что значение этой величины всегда представляет собой вещественное число.

Суть алгоритма: Если нужно записать "0", то необходимо целую часть величины $L(BLOCK)$ сделать четным числом, если нужно записать "1", то необходимо целую часть величины $L(BLOCK)$ сделать нечетным числом.

Пусть $X = \lfloor L(BLOCK) \rfloor$ и нам необходимо внедрить бит b_i. Если $|X| \bmod 2 = b_i$, то мы оставляем величину $L(BLOCK)$ без изменений. Если же $|X| \bmod 2 \neq b_i$, то мы уменьшаем число X на единицу, соответсвенно, величина $L(BLOCK)$ так же уменьшается на единицу.

Для уменьшения яркости блока пикселей на единицу, т.е. для изменения величины $L(BLOCK)$ на величину $\widetilde{L}(BLOCK) = L(BLOCK) - 1$, необходимо уменьшить на единицу все байты, отвечающие за красную, зеленую и синюю компоненты цвета каждого пикселя, принадлежащего данному блоку. Это следует из цепочки равенств:

$\widetilde{L}(BLOCK) = \{R(P_{i,j}) = R(P_{i,j}) - 1, G(P_{i.j}) = G(P_{i,j}) - 1, B(P_{i,j}) = B(P_{i,j}) - 1 \, \forall i = \overline{1..8} \, \forall j = \overline{1..8}\} = \frac{\sum_{i=1}^{8}\sum_{j=1}^{8} 0.299*(R(P_{i,j})-1)+0.587*(G(P_{i,j})-1)+0.114*(B(P_{i,j})-1)}{64} = \frac{\sum_{i=1}^{8}\sum_{j=1}^{8} L(P_{i,j})}{64} - 1 = L(BLOCK) - 1.$

Таким образом получаем, что в каждые 192 байта информации можно сохранить 1 бит. Например, в изображение $1024 * 768$ можно включить 1.5 килобайта информации, чего более чем достаточно для некоторого ЦВЗ. Для того, чтобы ЦВЗ не исказился при операции JPEG – BMP следует внедрять ЦВЗ, закодированный некоторым помехоустойчивым кодом C.

4 Краткое сравнение с другими алгоритмами.

Подавляющее большинство стеганографических алгоритмов, устойчивых к сжатию JPEG, могут после сжатия извлечь ЦВЗ только из файлов формата JPEG, тогда как описанному в данной статье алгоритму требуется уметь работать лишь с файлами формата BMP, что значительно проще.

Результаты сравнения объема информации, которые различные алгоритмы могут внедрить в фиксированное изображение, достаточно сильно друг от друга отличаются. К примеру, алгоритм Patchwork, описанный в [3], позволяет внедрить всего 1 бит на 20000 пикселей. Этот факт ставит под сомнение практическое использование данного алгоритма для изображений небольшого размера. Алгоритм, описанный в [4] позволяет внедрить 1 бит информации в 25 пикселей изображения, но рекомендует предварительно закодировать внедряемую информацию.

Список литературы

[1] В.Г.Грибунин,И.Н.Оков,И.В.Туринцев,Цифровая стеганография,Солон-Пресс, Москва, 2002.

[2] http://www.jpeg.org/jpeg/index.html.

[3] Bender W., Gruhl D., Morimoto N., Lu A. Techniques for Data Hiding, IBM Systems Journal. 1996. Vol. 35.

[4] Kutter M., Jordan F., Bossen F. Digital signature of color images using amplitude modulation, Proc. of the SPIE Storage and Retrieval for Image and Video Databases V. 1997. Vol. 3022. P. 518–526.

УДК 517.958:535.14

СРАВНЕНИЕ КОНСЕРВАТИВНЫХ СХЕМ И СХЕМ СУММАРНОЙ АППРОКСИМАЦИИ ДЛЯ УРАВНЕНИЯ ШРЕДИНГЕРА В АКСИАЛЬНО-СИММЕТРИЧНОЙ СРЕДЕ

© 2008 г. О. В. Матусевич

omatusevich@cs.msu.su

Лаборатория математического моделирования в физике

1 Введение

Рассмотрение различных оптических эффектов часто приводит к нелинейным уравнениям, для которых невозможно получить аналитическое решение. Часто для нахождения решений таких уравнений используются схемы суммарной аппроксимации (схемы с расщеплением). Идея построения схем суммарной аппроксимации заключается в расщеплении исходного нелинейного оператора на два более простых: линейный дифференциальный оператор, учитывающий дисперсионные и (или) дифракционные эффекты и нелинейный оператор, описывающий влияние нелинейной среды на распространение импульсов. Вообще говоря, дисперсионные (дифракционные) эффекты действуют на импульсы одновременно, однако в методе расщепления предполагается, что на каждом малом участке среды они действуют независимо. Для интегрирования нелинейного оператора используются методы Рунге-Кутта, итерационный метод или метод фазовых экранов. Однако, несмотря на простоту реализации метода суммарной аппроксимации, схемы с расщеплением не обладают свойством консервативности. В ряде случаев это приводит к значительному искажению решения. Поэтому, как было отмечено в [1,4] для схем с расщеплением требуется очень аккуратный выбор шага. Однако, при использовании подходящего шага в схемах суммарной аппроксимации снижается эффективность их использования. Несмотря на то, что схемы с расщеплением не консервативны, они используются при компьютерном моделировании в большом числе работ [6-14]. В данной работе показано, что при выборе неподходящей сетки для численного моделирования результаты расчетов, полученные с помощью схем суммарной аппроксимации, могут быть ошибочными.

2 Постановка задачи

Как известно, распространение оптического излучения в среде с кубичной нелинейностью в аксиально-симметричном случае, описывается следующим нелинейным уравнением Шредингера (НУШ) [3,4]:

$$\frac{\partial A}{\partial z} + i\tilde{D}\Delta_r A + i\alpha A\left|A\right|^2 = 0, \quad 0 < r < L_r, \quad 0 < z \leq L_z. \tag{1}$$

Здесь z - нормированная на дифракционную длину пучка основной волны продольная координата, $L = 2ka^2$, k – волновое число, a – начальный радиус пучка, $\Delta_r = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial}{\partial r}\right)$ – оператор Лапласа по координате r, измеряемой в единицах a, D – коэффициент дифракции. В выбранной нормировке он равен 1, но оставлен в (1) для удобства проведения расчетов. $alpha$ – коэффициент, характеризующий отношение начальной мощности импульса к характерной мощности самовоздействия, $A(r,z)$ – комплексная амплитуда пучка, нормированная на квадратный корень из максимальной интенсивности оптического излучения на входе в нелинейную среду. L_z – безразмерная длина нелинейной среды, L_r – ее поперечный размер.

На входе в нелинейную среду задается начальное распределение импульса:

$$A(r, z = 0) = A_0(r), \quad 0 \leq r \leq L_r. \tag{2}$$

Граничные условия для уравнения (1) имеют вид:

$$r\frac{\partial A}{\partial r}\,|_{r\to 0} = 0\,, \quad A\,|_{r=L_r} = 0\,. \tag{3}$$

В процессе моделирования необходимо контролировать выполнение следующих законов сохранения:

$$E = \int\limits_0^{L_r} |A|^2\, r dr, \quad H = \int\limits_0^{L_r} \left(-\tilde{D}\left|\frac{\partial A}{\partial r}\right|^2 + \alpha\,|A|^4 \right) r dr. \tag{4}$$

3 Разностные схемы

При построении разностной схемы для нахождения решений задачи (1) необходимо, чтобы для разностных аналогов решений выполнялись законы сохранения энергии и гамильтониана в разностном виде. Ниже будет построена консервативная схема для задачи (1). Для построения разностной схемы введем в области $\Omega = \Omega_r \times \Omega_z = \{0 \le r \le L_r\} \times \{0 \le z \le L_z\}$ сетку $\omega = \omega_r \times \omega_z$:

$$\omega_r = \{r_k = (k+0.5)h_r + ((k+0.5)h_r)^3\,, k = \overline{0, N_r}, h_r = L_r/(N_r + 0.5)\},$$

$$\omega_z = \left\{z_n = nh, n = \overline{0, N_z}, h = L_z/N_z\right\}.$$

Определим сеточную функцию A_h на ω и введем также следующие безындексные обозначения:

$$u = A_h(r_k, z_n), \quad \hat{u} = A_h(r_k, z_{n+1}),$$

$$\overset{0.5}{u} = 0.5(u + \hat{u}),$$

$$|\overset{0.5}{u}|^2 = 0.5(|\hat{u}|^2 + |u|^2),$$

$$\left(\overset{0.5}{u}\right)^2 = 0.5(\hat{u}^2 + u^2).$$

Оператор Лапласа по поперечной координате во внутренних узлах сетки аппроксимируется следующим образом [2,5]:

$$\Lambda_r w = \frac{1}{r_k(r_{k+1} - r_{k-1})}\left[(r_{k+1} + r_k)\frac{w(r_{k+1}, z_n) - w(r_k, z_n)}{r_{k+1} - r_k} - (r_k + r_{k-1})\frac{w(r_k, z_n) - w(r_{k-1}, z_n)}{r_k - r_{k-1}}\right],$$

где w - одна из сеточных функций u, $\hat{u}$.

Схема 1. Используя введенные обозначения, для уравнения (1) запишем во внутренних узлах сетки следующую разностную схему [2]:

$$\frac{\hat{u} - u}{h_z} + i\tilde{D}\Lambda_r \overset{0.5}{u} = f(\overset{0.5}{u}) \equiv -i\alpha \overset{0.5}{u}\,|\overset{0.5}{u}|^2, \tag{5}$$

Разностные уравнения (5) необходимо дополнить следующими условиями в соответствующих граничных точках сетки:

$$\begin{array}{l} u_{N_r,n} = 0,\ n = 0...N_z, \\ \frac{\hat{u}_0 - u_0}{h_z} + i\tilde{D}\frac{\overset{0.5}{u_1} - \overset{0.5}{u_0}}{0.5h_r^2} = f(\overset{0.5}{u}), \end{array} \tag{6}$$

Начальное условие для сеточной функции u задаётся в виде:

$$u_{k,0} = A_0(r_k), \quad k = 0, ..., N_r. \tag{7}$$

Для разрешения этой схемы применим итерационный процесс:

$$\frac{\overset{s+1}{\hat{u}} - u}{h_z} + i\tilde{D}\Lambda_r \overset{s+1}{\underset{0.5}{u}} = f(\overset{s}{\underset{0.5}{u}}), s = 0, 1, 2... \tag{8}$$

В граничных точках итерационный процесс можно записать следующим образом:

$$\begin{gathered} \overset{s+1}{u}_{N_r,n} = 0,\ n = 0...N_z, \\ \frac{\overset{s+1}{\hat{u}_0} - \overset{s+1}{u_0}}{h_z} + i\tilde{D}\frac{\overset{s+1}{\underset{0.5}{u_1}} - \overset{s+1}{\underset{0.5}{u_0}}}{0.5h_r^2} = f(\overset{s}{\underset{0.5}{u}}). \end{gathered} \tag{9}$$

Значения функций на нулевой итерации на верхнем слое вычисляются следующим образом:

$$\overset{s=0}{\hat{u}} = u. \tag{10}$$

Итерационный процесс прекращается при выполнении условия:

$$\max_{r_k} |\overset{s+1}{\hat{u}} - \overset{s}{\hat{u}}| \le \varepsilon \max_{r_k} |\overset{s}{\hat{u}}| + \delta. \tag{11}$$

Для нахождения неизвестной функции $\overset{s+1}{\hat{u}}$ используется метод прогонки. Доказательство консервативности схемы приводится, например, в [2].

Схема 2. Используя введенные ранее обозначения, определим значения функции на полуцелом слое:

$$z_{n+0.5} = (n + 0.5)h_z, \quad u_{n+0.5} = A_h(r_k, z_{n+0.5}),$$

$$\overset{0.5}{u^{(1)}} = 0.5(u_{n-0.5} + u_{n-1}), \quad \overset{0.5}{u^{(2)}} = 0.5(u_{n+0.5} + u_{n-0.5}), \quad \overset{0.5}{u^{(3)}} = 0.5(u_{n+1} + u_{n+0.5}).$$

Введем также дополнительную сетку $\tilde{\omega}_z^n = \{z_l = z_{n-0.5} + l\bar{h}_z, l = \overline{0, \bar{n}_z}, \bar{h}_z = h_z/\bar{n}_z\}$.

Разностную схему на основе метода расщепления можно записать так [1]:

$$\frac{u_{n-0.5} - u_{n-1}}{h_z} + i\tilde{D}\Lambda_r \overset{0.5}{u^{(1)}} = 0, \tag{12}$$

$$\frac{u_{n+0.5} - u_{n-0.5}}{2h_z} = -i\alpha \overset{0.5}{u^{(2)}} |\overset{0.5}{u^{(2)}}|^2, \tag{13}$$

$$\frac{u_{n+1} - u_{n+0.5}}{h_z} + i\tilde{D}\Lambda_r \overset{0.5}{u^{(3)}} = 0. \tag{14}$$

Уравнение (13) нелинейно, для его разрешения могут использоваться различные методы. В данной работе рассматривается метод Рунге-Кутта (РК) второго порядка:

$$u_{n,l+1}^{(2)} = u_{n,l}^{(2)} + \frac{\bar{h}_z}{2}\left[f(z_l, u_{n,l}^{(2)}) + f\left(z_l + \bar{h}_z, u_{n,l}^{(2)} + \bar{h}_z f\left(z_l, u_{n,l}^{(2)}\right)\right)\right], \quad l = 0...\bar{n}_z, \quad u_{n,0}^{(2)} = u_n^{(2)}. \tag{15}$$

и итерационный процесс (ИП):

$$\frac{\overset{s+1}{u_{n+0.5}} - \overset{s+1}{u_{n-0.5}}}{h_z} = 2f\begin{pmatrix} \overset{s}{\underset{0.5}{u^{(2)}}} \\ \end{pmatrix}, \tag{16}$$

где $s = 0, 1, 2, ...$ - номер итерации.

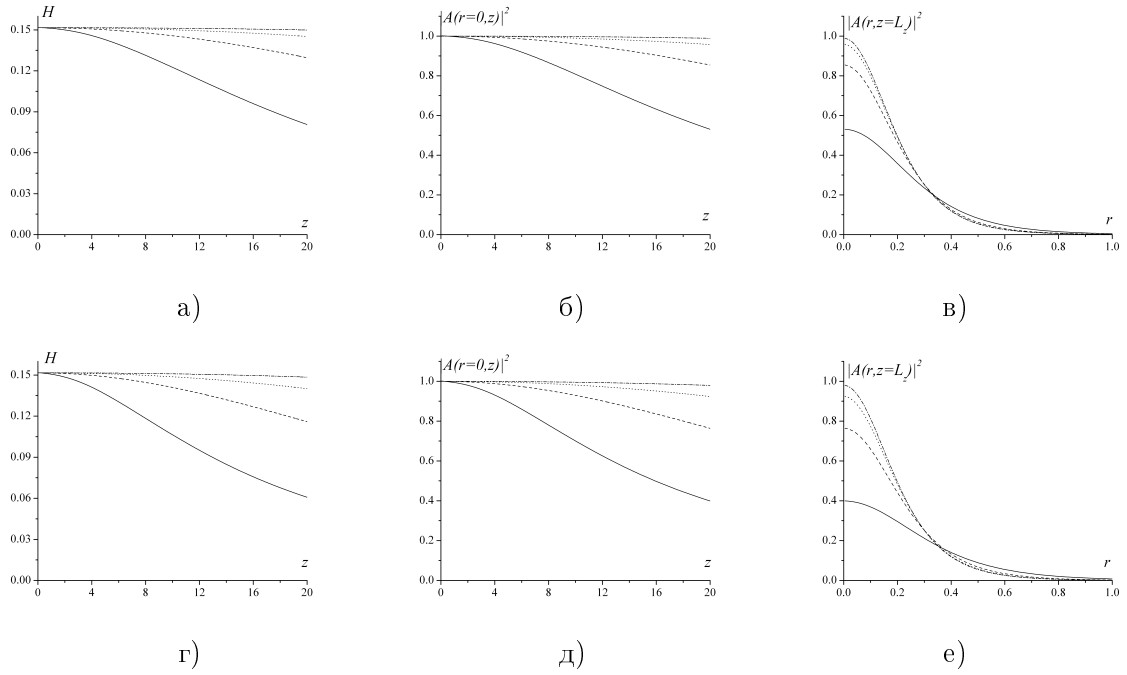


Рис. 1: Сравнение гамильтониана (а, г), пиковой интенсивности (б, д) и распределения интенсивности в конечном сечении среды (в, е) для различных значений N_z =2000 (сплошная линия), 5000 (пунктир), 10000 (точечная линия), 20000 (штрих-пунктир) для *Схемы 2* с использованием метода Рунге-Кутта (а, б, в) и итерационного процесса (г, д, е) для $\alpha = 5$, $D = 0.1$, $L_z = 20$ с начальным условием, имеющим форму таунсова солитона.

4 Результаты компьютерного моделирования

Как было сказано выше, схемы суммарной аппроксимации не обладают свойством консервативности. Это может приводить к искажению решений уравнения (1). Поскольку аналитическое решение уравнения (1) в общем случае неизвестно, начальное условие выбиралось в виде пучка, для которого заведомо известно его поведение при распространении в среде с керровской нелинейностью: таунсов солитон. Поскольку солитон распространяется без изменения своей формы, то при возникновении искажений в форме решения, полученного с помощью одной из разностных схем, можно говорить о некорректной работе этой схемы. Для нахождения солитонных решений уравнения (1) использовался метод, описанный в работах [15, 16].

При компьютерном моделировании было показано, что для нахождения верного решения уравнения (1) схемами суммарной аппроксимации (*Схема 2*) следует очень аккуратно подходить к выбору шага по продольной координате. Рис.1 иллюстрирует, как влияет шаг сетки по координате z на нахождение решения *Схемой 2*.

Как из него видно, решения, полученные *Схемой 2* при шаге по продольной координате большем 0.002, существенно отличаются от истинного солитона (Рис.1б,в,д,е). При этом существенно искажается и гамильтониан H (Рис.1а,г). Это говорит о том, что при использовании *Схемы 2* необходимо выбирать очень мелкий шаг по продольной координате. Необходимо подчеркнуть также, что при использовании *Схемы 1* можно существенно сократить число точек по продольной координате по сравнению со *Схемой 2* без потери качества решения. Так, на Рис.2 показана эволюция таунсова солитона на трассе $z < 20$, рассчитанная с помощью *Схемы 1* с продольным шагом равным 0.01. Из рисунка можно заключить, что интенсивность решения, полученного с помощью консервативной схемы, не изменяется вдоль координаты z, т.е. *Схема 1* работает корректно.

Далее рассмотрим, как влияет увеличение длины трассы L_z при фиксированном продольном шаге $h_z = 0.006$ на решения, полученные схемами суммарной аппроксимации. С этой целью на Рис.3 изображена эволюция пиковой интенсивности и гамильтониана решений урав-

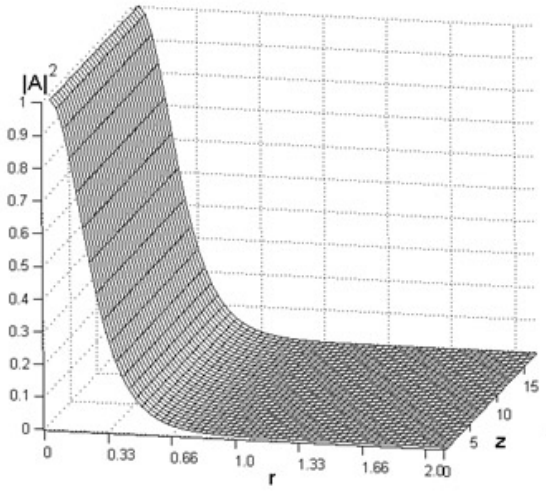


Рис. 2: Решение уравнения (1), найденное с помощью *Схемы 1*, на трассе $L_z = 20$ с числом точек по координате z равным N_z =2000 для $\alpha = 5$, $D = 0.1$, $L_z = 20$ с начальным условием, имеющим форму таунсова солитона.

нения (1) для различных значений L_z. Результаты расчетов для консервативной схемы и схем с расщеплением несущественно отличаются друг от друга при $L_z \leq 20$ (Рис.3,I). Однако, при увеличении трассы до 40 (Рис.3,II) становится заметно, что пиковая интенсивность и гамильтониан, полученные *Схемой 2*, начинают отклоняться от своих константных значений. Дальнейший рост длины трассы (Рис.3,III) приводит к значительному расхождению расчетов схем суммарной аппроксимации с истинным решением.

При этом важно подчеркнуть, что увеличение длины трассы не влияло на решение, получаемое консервативной схемой. Проведенное для различных значений h_z компьютерное моделирование позволяет сделать вывод о том, что для схем суммарной аппроксимации для любого фиксированного шага по координате z всегда можно так подобрать длину нелинейной среды, что решение будет существенно отклоняться от своего истинного значения. Необходимо отметить еще одну особенность схем суммарной аппроксимации: при расчетах на больших трассах возможно также существенное искажение решения при достижении решением границ поперечной области. Этот случай демонстрирует Рис.4.

5 Выводы

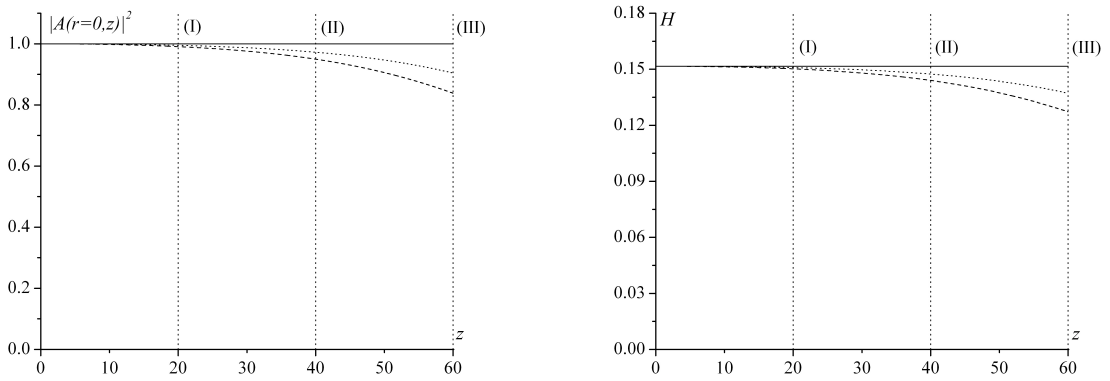
В данной работе было проведено сравнение широко используемых схем суммарной аппроксимации с симметричной консервативной схемой для задачи распространения оптического излучения в аксиально-симметричной среде. Выше были показаны границы применимости для схем суммарной аппроксимации, а также преимущества использования консервативных схем.

Список литературы

[1] И. Г. Захарова, Ю. Н. Карамзин, В. А. Трофимов. *Метод расщепления в задачах нелинейной оптики.* Изд-во ИПМ им. М.В. Келдыша АН СССР, Москва, 1989.

[2] Ю. Н. Карамзин, А. П. Сухоруков, В. А. Трофимов. *Математическое моделирование в нелинейной оптике.* Изд-во Московского Университета, Москва, 1989.

[3] С. А. Ахманов, В. А. Выслоух, А. С. Чиркин. *Оптика фемтосекундных лазерных импульсов.* М.: Наука, 1988.



а) б)

Рис. 3: Эволюция пиковой интенсивности (а) и гамильтониана (б) решения уравнения (1), полученного с помощью *Схемы 1* (сплошная линия) и *Схемы 2* с использованием метода Рунге-Кутта (точечная линия) и итерационного процесса (пунктир) при фиксированном шаге по продольной координате $h_z = 0.0006$ для $\alpha = 5$, $D = 0.1$ для различной длины среды L_z= 20 (I), 40 (II) и 60 (III) с начальным условием, имеющим форму таунсова солитона.

[4] Г. Агравал. *Нелинейная волоконная оптика.* М.: Мир, 1996.

[5] А. А. Самарский, В. Б. Андреев. *Разностные методы для эллиптических уравнений.* М.: Наука, 1976.

[6] M. D. Feit, J. A. Fleck, Jr. A. Steiger. *Solution of the Schrodinger equation by a spectral method.* J.Comput.Phys. 1982. V.47. N.3. P.412-433.

[7] J. A. Fleck, J. R. Morris, M. D. Feit. *Time-dependent propagation of high energy laser beams through the atmosphere.* Appl. Phys. 1976. V.10. N.2. P.129-160.

[8] T. R. Taha, M. J. Ablowitz. *Analytical and numerical aspects of certain nonlinear evolution equation.* J. Comput. Phys. 1984. V.55. N.2. P.203-230.

[9] S. Ashihara, J. Nishina, T. Shimura, K. Kuroda. *Soliton compression of femtosecond pulses in quadratic media.* JOSA B. 2002. V.19. N.10. P.2505-2510.

[10] A. V. Buryak, P. D. Trapani, D. V. Skryabin, S. Trillo. *Optical solitons due to quadratic nonlinearities: From basic physics to futuristic application.* Phys. Rep. 2002. V.370. P.63-235.

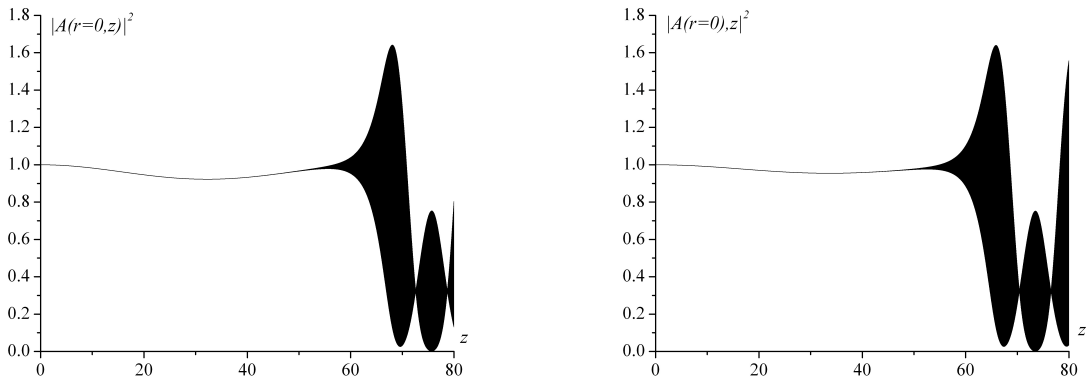
[11] D. Salerno et al. *Noise-seeded spatiotemporal modulation instability in normal dispersion.* Phys. Rev. E. 2004. V.70. N.6. P.065603-1- 065603-4.

[12] Yu. S. Kivshar, D. E. Pelinovsky. *Self-focusing and transverse instabilities of solitary waves.* Phys. Rep. 2000. V.331. P.117-195.

[13] B. A. Malomed, A. A. Nepomnyashchy. *Modulational instability of an axisymmetric state in a two-dimensional Kerr Medium.* Phys. Rev. E. 1995. V.52. N.1. P.1238-1240.

[14] L. Berge et al. *Self-focusing and soliton-like structures in material with competing quadratic and cubic nonlinearity.* Phys. Rev. E. 1997. V.55. N.3. P.3555-3570.

[15] С. А. Варенцова, В. А. Трофимов. *О разностном методе нахождения собственных мод нелинейного уравнения Шредингера.* Вестн. Моск. Ун-Та. Сер.15. 2005. N.3.С.16-22.



а) б)

Рис. 4: Эволюция пиковой интенсивности решения уравнения (1), полученного с помощью *Схемы 2* с использованием метода Рунге-Кутта (а) и итерационного процесса (б) при достижении решением границ поперечной области для следующих значений параметров: $\alpha = 5$, $D = 0.1$, $L_z = 80$ (I), $h_z = 0.002$, $L_r = 2$ с начальным условием, имеющим форму таунсова солитона.

[16] О. В. Матусевич, В. А. Трофимов. *Итерационный метод нахождения собственных функций системы двух уравнений Шредингера с комбинированной нелинейностью.* ЖВМиМФ. 2008. Т. 48. №4.

УДК 519.7

О СЛОЖНОСТИ УМНОЖЕНИЯ ПОЛИНОМОВ И МАТРИЦ

© 2008 г. А. Д. Поспелов

alexey.pospelov@gmail.com

Кафедра математической кибернетики

1 Введение

Сложность и эффективные алгоритмы решения математических задач являются важными объектами исследования современной математики. Построение быстрых алгоритмов решения алгебраических задач является предметом изучения *вычислительной алгебры*. *Алгебраическая теория сложности* расширяет предмет поиском нижних оценок.

Алгебраическая теория сложности изучает сложность *алгебраических* задач в *алгебраических* моделях вычислений. Примерами алгебраических задач являются умножение чисел, заданных в позиционной системе счисления, нахождение наибольшего общего делителя двух натуральных чисел, умножение полиномов, заданных векторами коэффициентов, вычисление значения полинома в заданной точке, умножение матриц, нахождение определителя квадратной матрицы, поиск корней характеристического многочлена квадратной матрицы, обращение невырожденной матрицы, решение системы линейных алгебраических уравнений и т. д. Нетрудно видеть, что каждый из вышеприведённых примеров подразумевает набор входных переменных и набор функций от входных переменных, которые необходимо вычислить. Алгебраическая модель вычислений подразумевает пошаговую вычислительную процедуру, принимающую на вход конечный набор алгебраических величин и вычисляющую на каждом шаге определённую функцию от входа и результатов предыдущих шагов. При этом результат очередного шага, как правило, может быть использован последующими шагами в качестве аргумента соответствующей функции неограниченное число раз. Формализация описанной пошаговой процедуры называется обычно *алгебраическим алгоритмом*, а число шагов, необходимое и достаточное для вычисления выходных функций, называется *сложностью алгебраического алгоритма*. Минимальная сложность алгебраического алгоритма, решающего данную алгебраическую задачу, называется *сложностью алгебраической задачи*.

Обычно фиксируется некоторый подкласс алгебраических алгоритмов, относительно которого изучается вопрос о сложности задачи. Часто ограничивается множество функций, используемых в шагах алгоритма, а также разрешённые аргументы для этих функций из числа входов задачи и промежуточных результатов.

Одной из важнейших задач алгебраической теории сложности является сложность умножения многочленов. Задача заключается в вычислении вектора коэффициентов многочлена-произведения по векторам коэффициентов многочленов-множителей. Многочлены могут рассматриваться как над полями, так и над кольцами, быть как многочленами одного переменного, так и нескольких. При этом на каждом шаге алгоритма разрешается использовать сложение, вычитание, умножение или деление, аргументами которых могут являться входные коэффициенты, результаты предыдущих шагов или константы из поля. Работа алгоритма считается завершённой, если каждый коэффициент многочлена-произведения вычислен на некотором шаге алгоритма.

Пусть даны многочлены

$$a(X) = \sum_{\mu=0}^{m} a_\mu X^\mu, \qquad b(X) = \sum_{\nu=0}^{n} b_\nu X^\nu$$

над полем k. Коэффициенты их произведения

$$a(X)b(X) = c(X) = \sum_{\lambda=0}^{m+n} c_\lambda X^\lambda$$

определяются формулами

$$c_\lambda = \sum_{\mu=\max\{0,\,\lambda-n\}}^{\min\{\lambda,\,m\}} a_\mu b_{\lambda-\mu}, \qquad \lambda = 0,\,\ldots,\,m+n. \tag{1}$$

Очевидно, что для вычисления всех коэффициентов c_λ согласно (1) требуется mn операций умножения и $O(mn)$ операций сложения, то есть всего $O(mn)$ арифметических операций.

Известно, что данная схема не является оптимальной. Впервые это было установлено в [7], где был построен алгоритм умножения чисел в двоичном представлении, имеющий сложность $O(n^{\log_2 3})$ (алгоритм Карацубы) для умножения двух чисел длины n, непосредственно трансформирующийся в алгоритм умножения полиномов степени $n-1$ над произвольным полем, имеющий такую же сложность по числу арифметических операций. Идея алгоритма заключается в умножении двух полиномов степени 2 с использованием не четырёх (согласно (1)), а трёх операций умножения, и применении техники разделяй-и-властвуй.

В [10] алгоритм Карацубы был обобщён Тоомом, предложившим метод построения алгоритма умножения чисел длины n, имеющий арифметическую сложность $O(n^{1+\varepsilon})$ для любого наперёд заданного положительного ε, содержащий алгоритм той же сложности для умножения полиномов степени n над произвольным полем. Алгоритм Тоома основан на том, что полином степени n можно задавать как вектором его $n+1$ коэффициентов

$$(a_0,\,a_1,\,\ldots,\,a_n) \tag{2}$$

так и вектором его значений в $n+1$ различных точках

$$(a(X_0),\,a(X_1),\,\ldots,\,a(X_n)). \tag{3}$$

При этом во втором представлении умножение двух многочленов осуществляется тривиально — достаточно лишь вычислить значения обоих полиномов в $2n+1$ различных точках и попарно перемножить их; результирующий вектор и будет вектором значений полинома-произведения в выбранных $2n+1$ точках. Эффективный способ перехода от вектора коэффициентов к вектору значений основан на схеме Горнера вычисления значения полинома в точке, а от вектора значений к вектору коэффициентов — на менее эффективной, но, тем не менее, не ухудшающей порядок сложности $O(n^{1+\varepsilon})$ интерполяционной формуле Лагранжа.

Эти идеи были развиты и обобщены Шёнхаге и Штрассеном в [26], предложившим алгоритм сложности $O(n\log n\log\log n)$ умножения двух чисел длины n в двоичном представлении, содержащий алгоритм той же сложности для умножения полиномов степени n над произвольным полем. В течение 36 лет этот алгоритм оставался самым быстрым, однако Фюрер в 2007 году, используя подходы, описанные Шёнхаге и Штрассеном в [26], и комбинируя их несколько иначе, предложил в [20] алгоритм умножения чисел длины n в двоичном представлении (и полиномов степени n над произвольным полем), имеющий сложность $n\log n 2^{O(\log^* n)}$.[1] На сегодняшний день этот алгоритм является лучшим по сложности алгоритмом умножения полиномов над произвольным полем.

Одновременно с этим известны более эффективные алгоритмы умножения полиномов над полями, удовлетворяющими некоторым дополнительным условием. Существует алгоритм перехода от представления (2) к представлению (3), работающий в случае, когда поле k, над которым рассматриваются многочлены, содержит ровно $n+1$ различных корней из 1 (см., например, [15]). Этот алгоритм, имеющий сложность $O(n\log n)$, известен как *быстрое преобразование Фурье*, а лежащее в его основе преобразование — как *дискретное преобразование Фурье*. Условию наличия необходимых корней из 1 удовлетворяет, например, поле комплексных чисел (для любого n) или любое алгебраически замкнутое поле характеристики p (в случае, когда $p \nmid n$). Обратное преобразование имеет такую же сложность, таким образом, над такими

[1] Напомним, что функция $\log^* n$ натурального аргумента n, определяется как наибольшее ℓ, для которого определён $\underbrace{\log\ldots\log}_{\ell\text{ раз}} n$.

полями k можно строить алгоритмы умножения полиномов степени n, имеющие сложность $O(n \log n)$.

Лучшая нижняя оценка сложности умножения полиномов степени n над произвольным полем равна $2n-1$ (см., например, [11]). В действительности, эта оценка получена для числа активных *умножений* любого алгоритма, вычисляющего произведение двух полиномов степени n. Однако, для произвольного поля и без ограничений вида алгоритма более высоких нижних оценок пока не получено.

В случае конечного поля известна следующая оценка, принадлежащая Каминскому (см. [21]):

$$C_q(n) \geq \left(3 + \frac{(q-1)^2}{q^5 + (q-1)^3}\right) n - o(n),$$

где $C_q(n)$ — сложность *билинейного*[2] алгоритма умножения полиномов степени n над полем, состоящем из q элементов.

Бюргиссер и Лотц рассмотрели в [16] сложность умножения полиномов в модели вычислений, допускающей использование в построении алгоритма умножения коэффициенты, ограниченные некоторой фиксированной константой. В этом случае получена нижняя оценка $\Omega(n \log n)$ сложности умножения полиномов порядка n над полем комплексных чисел или любым его подполем (например, полями рациональных, вещественных или алгебраических чисел). Эта оценка справедлива, в действительности, для числа аддитивных операций алгоритма.

Существует гипотеза о том, что сложность умножения полиномов степени n над произвольными полями равна $O(n \log n)$ (см. [15]).

Пусть теперь даны многочлены

$$a(X_1, \ldots, X_s) = \sum_{\mu_1=0}^{m_1} \cdots \sum_{\mu_s=0}^{m_s} a_{(\mu_1, \ldots, \mu_s)} X_1^{\mu_1} \cdots X_s^{\mu_s},$$

$$b(X_1, \ldots, X_s) = \sum_{\nu_1=0}^{n_1} \cdots \sum_{\nu_s=0}^{n_s} b_{(\nu_1, \ldots, \nu_s)} X_1^{\nu_1} \cdots X_s^{\nu_s}$$

над некоторым полем k, и пусть $a_{(m_1, \ldots, m_s)}$, $b_{(n_1, \ldots, n_s)} \neq 0$. Заметим, что степень a, определяемая как максимальная степень монома[3] в a, равна $m_1 \cdots m_s$, а степень b равна $n_1 \cdots n_s$. Их произведение

$$c(X_1, \ldots, X_s) = \sum_{\lambda_1=0}^{m_1+n_1} \cdots \sum_{\lambda_s=0}^{m_s+n_s} c_{(\lambda_1, \ldots, \lambda_s)} X_1^{\lambda_1} \cdots X_s^{\lambda_s}$$

имеет степень $(m_1+n_1) \cdots (m_s+n_s)$. Коэффициенты c определяются формулами

$$c_{(\lambda_1, \ldots, \lambda_s)} = \sum_{\substack{0 \leq \mu_1 \leq m_1,\ 0 \leq \nu_1 \leq n_1:\ \mu_1+\nu_1=\lambda_1 \\ \ldots \\ 0 \leq \mu_s \leq m_s,\ 0 \leq \nu_s \leq n_s:\ \mu_s+\nu_s=\lambda_s}} a_{(\mu_1, \ldots, \mu_s)} b_{(\nu_1, \ldots, \nu_s)}, \quad 0 \leq \lambda_\sigma \leq m_\sigma + n_\sigma,\ 1 \leq \sigma \leq s. \quad (4)$$

Нетрудно видеть, что для вычисления всех $c_{(\lambda_1, \ldots, \lambda_s)}$ непосредственно по формулам (4) требуется $O((m_1+n_1) \cdots (m_s+n_s))$ арифметических операций.

Указанный способ вычисления произведения многочленов многих переменных не является оптимальным. Пан в [24] предложил способ сведения умножения полиномов нескольких

[2]Формальное определение билинейного алгоритма будет дано в разделе 2, однако заметим, что билинейные алгоритмы представляют собой естественную модель вычисления произведений в алгебрах: большинство задач имеют оптимальными билинейные алгоритмы, а почти все известные нижние оценки сложности умножения получаются для билинейных алгоритмов.

[3]Степенью монома называется, как обычно, сумма степеней всех входящих в моном переменных.

переменных степени N к умножению полиномов одного переменного степени $O(N)$, позволяющий получить алгоритмы умножения сложности, не превосходящей $O(N \log N \log\log N)$, над произвольным полем и сложности, не превосходящей $O(N \log N)$, над полем, поддерживающим быстрое преобразование Фурье. Метод Пана основан на взаимно однозначной кодировке мономов полинома многих переменных, сохраняемой умножением полиномов.

В данной статье мы приведём простой метод построения быстрых алгоритмов умножения полиномов многих переменных, имеющих такую же сложность, как и построенные методом Пана. Новый метод основан на изоморфном вложении алгебры полиномов многих переменных в должным образом подобранную коммутативную групповую алгебру и эффективном алгоритме умножения в групповой алгебре.

Центральной задачей современной алгебраической теории сложности является сложность умножения матриц. Пусть

$$a = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{pmatrix}, \qquad b = \begin{pmatrix} b_{11} & \dots & b_{1s} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{ns} \end{pmatrix}$$

— матрицы над полем k. Их произведение

$$c = \begin{pmatrix} c_{11} & \dots & c_{1s} \\ \dots & \dots & \dots \\ c_{m1} & \dots & c_{ms} \end{pmatrix}$$

определяется формулами

$$c_{\mu\sigma} = \sum_{\nu=1}^{n} a_{\mu\nu} b_{\nu\sigma}, \qquad 1 \le \mu \le m,\ 1 \le \sigma \le s. \tag{5}$$

Для вычисления коэффициентов матрицы c по формулам (5) необходимо совершить mns умножений и $m(n-1)s$ сложений в k. Известно, что данная процедура не является оптимальной. Впервые это было установлено Штрассеном в [28], предложившим алгоритм сложности $O(n^{\log_2 7})$ умножения квадратных матриц порядка n над произвольным полем. Алгоритм Штрассена основан на конструкции умножения матриц размера 2×2 с использованием 7, а не 8 умножений, а также на применении техники разделяй-и-властвуй. В течение 9 последующих лет эта верхняя оценка сложности оставалась наилучшей. В 1978 году Пан в (см. [23]) смог впервые несколько улучшить эту оценку, и в последующие 9 лет были предложены различные методы и алгоритмы умножения матриц, лучший из которых — алгоритм Копперсмита-Винограда (см. [19]), предложенный впервые в 1987 году (опубликованный, однако, несколько позже) и имеющий сложность, не превосходящую $O(n^{2,376})$, для умножения матриц порядка n, — остаётся лучшим по сложности на сегодняшний день.

При получении верхних оценок сложности умножения матриц была разработана естественная вычислительная модель для различных задач линейной алгебры. Было предложено несколько подходов, каждый из которых позволял улучшить текущую верхнюю оценку сложности умножения матриц. В числе наиболее значительных — *приближённые* (или параметризованные «λ-») алгоритмы, *частичное умножение матриц*, умножение нескольких независимых пар матриц одним алгоритмом (*теорема о прямой сумме* или *τ-теорема*) и *относительные вычисления* (см. [1]). Все эти подходы были сформулированы независимо, с использованием различных базовых понятий и основываясь на практически не связанных соображениях.

Большинство существующих нижних оценок сложности умножения матриц справедливо в действительности для числа *умножений*, необходимых для вычисления произведения двух матриц. Долгое время лучшей оценкой была оценка $2n^2 - 1$ умножений для вычисления произведения двух матриц порядка $n \times n$ (см., например, [11, 15]). Однако Блезеру в [12] удалось поднять эту оценку до

$$\frac{5}{2} n^2 - 3n \tag{6}$$

для сложности умножения матриц порядка $n \times n$. Для маленьких значений $n \geq 3$ в [14] получена оценка

$$2n^2 + n - 2, \tag{7}$$

превосходящая (6) для $n \leq 7$. Оценка Блезера справедлива для сложности умножения матриц над произвольными полями. Над конечными полями Шпилькой в [27] получена более высокая нижняя оценка: $3n^2 - o(n^2)$ для поля $GF(2)$ и $\left(\frac{5}{2} + \frac{3}{2(p^3-1)}\right)n^2 - o(n^2)$ для поля $GF(p)$. На сегодняшний день оценки Блезера и Шпильки являются лучшими нижними оценками сложности умножения матриц.

Ран Раз в [25] получил первую нелинейную нижнюю оценку сложности умножения матриц в ограниченной модели вычислений. Если в алгоритмах разрешено использование лишь коэффициентов, ограниченных по модулю некоторой заранее фиксированной константой, то арифметическая сложность умножения матриц не может быть меньше, чем $\Omega(n^2 \log n)$ для умножения двух матриц порядка $n \times n$. Эта оценка справедлива, в действительности, и для числа аддитивных операций алгоритма.

Известно, что минимальное число умножений, достаточное для вычисления произведения двух матриц порядка 2×2 равно 7 (см., например, [15]). Из оценки (7) следует, что для вычисления произведения двух матриц порядка 3×3 необходимо 19 умножений, а из алгоритма Ладермана (см. [22]) следует, что достаточно 23 умножений. На сегодняшний день неизвестно, чему равно минимальное число умножений, достаточное для вычисления произведения двух матриц порядка 3×3.

В 2003 году Коэн и Уманс в [17] предложили *теоретико-групповой подход* к сложности умножения матриц. Этот подход основан на известном факте теории представлений (см. [5, 6]) о том, что любое линейное представление над полем комплексных чисел конечной группы изоморфно алгебре квадратных матриц некоторого порядка. Вместе с теоремой о прямой сумме это даёт эффективный способ получения верхних оценок сложности умножения матриц. В [18] были получены первые нетривиальные верхние оценки, основанные на теоретико-групповом подходе. Эти верхние оценки представляют собой старые конструкции, записанные на новом теоретико-групповом языке. Несмотря на то, что оценка Копперсмита-Винограда в [18] улучшена не была, появление нового подхода и верхних оценок, полученных на основе него, стимулировало новую волну интереса к сложности умножения матриц, так как позволило систематизировать и единообразно изложить большинство предыдущих верхних оценок.

Из модели вычислений, построенной в [17], вытекает, что решение вопроса о сложности умножения в *групповых алгебрах*, исчерпывающе решает задачу сложности умножения матриц. Этот факт стимулировал рост интереса к сложности умножения в групповых алгебрах. В [4, 3] была разработана техника описания структуры групповых алгебр и модель получения нижних оценок сложности. Там же была изучена групповая алгебра подстановок третьего порядка над полем комплексных и вещественных чисел, получена её структура и сложность умножения. В [2] разработанные методы были применены для изучения групповой алгебры симметрий квадрата над полем комплексных чисел, для установления её структуры и сложности умножения в ней. В [8, 9] были изучены произвольные коммутативные алгебры над различными полями. В [9] была получена структура и сложность коммутативных групповых алгебр над полями вещественных и комплексных чисел, а в [8] этот результат был обобщён также на алгебры над произвольными алгебраически замкнутыми полями. В действительности, результат [8] справедлив также над полями, содержащими лишь всевозможные корни определённой степени из единицы.

В данной статье посредством анализа одной групповой алгебры мы покажем новый способ установления сложности умножения матриц порядка 3×3. В действительности, мы рассмотрим наименьшую групповую алгебру, содержащую алгебру матриц порядка 3×3 как изоморфную подалгебру, имеющую относительно простую структуру и покажем, что любое улучшение нижней или верхней оценки сложности умножения в этой алгебре улучшает нижнюю (соответственно, верхнюю) оценку сложности умножения матриц 3×3. Мы сформулируем и докажем альтернативу о билинейной сложности умножения в алгебре матриц третьего порядка, алгебре чётных подстановок четвёртого порядка и гипотезе о прямой сумме.

2 Основные понятия

Будем называть *алгеброй над полем* k произвольное конечномерное линейное пространство над полем k с дополнительно заданной ассоциативной операцией умножения. Умножение в алгебре, как функция двух векторов, линейно по обоим аргументам (иными словами, умножение является *билинейным отображением* в алгебре). Будем полагать, что в алгебре есть нейтральный элемент относительно умножения (единица). Если умножение коммутативно, то алгебру будем называть *коммутативной алгеброй*. *Базисом алгебры* будем называть любой базис соответствующего линейного пространства, а *размерностью алгебры* — размерность основного линейного пространства.

Алгебры A и B над полем k называются *изоморфными*, если существует изоморфизм φ линейных пространств A и B (для определенности пусть $\phi : A \to B$), для которого $\phi(ab) = \phi(a)\phi(b)$ для любых a, b из A. Очевидно, что размерности изоморфных алгебр равны, а свойства наличия единицы и коммутативности в изоморфных алгебрах одновременно либо выполняются, либо не выполняются.

Одним из способов построения больших алгебр из меньших является *прямое произведение*. Пусть A и B — алгебры над k. Прямым произведением A и B называется алгебра $C = A \times B$ над k, элементы которой являются парами элементов (a, b), где $a \in A$, $b \in B$. Сложение и умножение в C производится покомпонентно:

$$\alpha(a, b) + \beta(c, d) = (\alpha a + \beta c, \alpha b + \beta d), \qquad (a, b)(c, d) = (ac, bd).$$

Очевидно, $\dim A + \dim B = \dim C$.

Подмножество B алгебры A называется *подалгеброй алгебры* A, если соответствующее линейной пространство B является линейным подпространством A, и умножение в A элементов из B не выводит за пределы B: для любых b_1, $b_2 \in B$ их произведение $b_1 b_2$ также принадлежит B. Подалгебра I алгебры A называется *левым (правым) идеалом* A, если для любого $a \in A$ справедливо $aI = I$ (соответственно, $Ia = I$)[4]. Левый идеал, являющийся одновременно правым, называется *двусторонним*. Тривиальными левыми и правыми идеалами в A являются A и $\{0\}$. Очевидно, пересечение конечного числа левых (правых) идеалов является также левым (соответственно, правым) идеалом. Левый (правый) идеал I в A называется *максимальным*, если из $I \subsetneq J \subseteq A$, где J — идеал в A, следует, что $J = A$.

Пусть A — алгебра над полем k размерности n, и пусть $a_1, \ldots, a_n$ — некоторый базис A. Если множество $\{a_1, \ldots, a_n\}$ образует группу относительно умножения в A, то такой базис называется *групповым базисом*, а алгебра соответственно *групповой алгеброй*. Обратно, пусть $G = \{g_1, \ldots, g_n\}$ — группа порядка n, и k — поле. Тогда алгебра

$$B = \{\alpha_1 g_1 + \cdots + \alpha_n g_n | \alpha_\nu \in k,\ 1 \le \nu \le n\}$$

с умножением, определяемым по правилу

$$\left(\sum_{\nu=1}^{n} \alpha_\nu g_\nu\right)\left(\sum_{\mu=1}^{n} \beta_\nu g_\nu\right) = \sum_{\kappa=1}^{n}\left(\sum_{\nu,\,\mu:g_\nu g_\mu = g_\kappa} \alpha_\nu \beta_\mu\right) g_\kappa,$$

является групповой. Будем обозначать алгебру над k с групповым базисом, образующим группу G, через $k[G]$. Очевидно, что $k[G]$ коммутативна тогда и только тогда, когда G — абелева.

Нашей основной задачей является установление трудоемкости вычисления произведения элементов в алгебрах. С этой целью необходимо зафиксировать модель вычислений и функционал сложности. Общепринятыми в алгебраической теории сложности являются билинейная и мультипликативная модель вычисления с соответствующими сложностями (см. [15]).

Пусть U, V, W — конечномерные линейные пространства над полем k, и $\phi : U \times V \to W$ — билинейное отображение. *Билинейным вычислением (билинейным алгоритмом)* для ϕ

[4]Под произведением элемента на множество понимается множество произведений элемента на все элементы множества.

называется такая последовательность $(f_1, g_1, w_1, \ldots, f_r, g_r, w_r)$, где $f_\rho \in U^*$, $g_\rho \in V^*$, $w_\rho \in W$, $1 \le \rho \le r$, что для любых $u \in U$, $v \in V$

$$\phi(u, v) = \sum_{\rho=1}^{r} f_\rho(u) g_\rho(v) w_\rho.$$

r называется *длиной* билинейного вычисления. *Рангом* или *билинейной сложностью* ϕ называется длина кратчайшего билинейного вычисления для ϕ. Ранг ϕ обозначается $\operatorname{rk}\phi$.

Ранг умножения в алгебре A (являющегося билинейным отображением $A \times A \to A$) называется *рангом алгебры* A и обозначается $\operatorname{rk} A$.

Обобщением билинейного вычисления и ранга является квадратичное вычисление и мультипликативная сложность. *Квадратичным вычислением (квадратичным алгоритмом)* для ϕ является такая последовательность $(f_1, g_1, w_1, \ldots, f_\ell, g_\ell, w_\ell)$, где $f_\lambda, g_\lambda \in (U \times V)^*$, $w_\lambda \in W$, $1 \le \lambda \le \ell$, что для любых $u \in U$, $v \in V$

$$\phi(u, v) = \sum_{\lambda=1}^{\ell} f_\lambda(u, v) g_\lambda(u, v) w_\lambda.$$

ℓ называется *длиной* квадратичного вычисления. *Мультипликативной сложностью* ϕ называется длина кратчайшего квадратичного вычисления для ϕ. Мультипликативная сложность ϕ называется $C(\phi)$.

Мультипликативная сложность умножения в алгебре A называется *мультипликативной сложностью алгебры* A и обозначается $C(A)$.

Очевидно, для любого ϕ справедливо $C(\phi) \le \operatorname{rk}\phi \le 2C(\phi)$.

Главным инструментом получения верхних оценок является конструкция алгоритма умножения в алгебре, имеющего сложность, не превосходящей требуемую верхнюю оценку. Главным инструментом получения нижних оценок мультипликативной и билинейной сложностей алгебр являются нижняя оценка Алдера-Штрассена и теорема Блезера, описывающая все алгебры, на которых достигается оценка Алдера-Штрассена.

Теорема 1 (А. Алдер, Ф. Штрассен, см. [11]). *Для произвольной ассоциативной алгебры A выполняется*

$$C(A) \ge 2 \dim A - t(A), \tag{8}$$

где $t(A)$ — число максимальных двусторонних идеалов A.

Отсюда, в частности, следует, что $\operatorname{rk} A \ge 2 \dim A - t(A)$. Алгебры, для которых оценка (8) совпадает с верхней, называются *алгебрами минимального ранга*.

Прежде чем описать структуру алгебр минимального ранга над произвольными полями, дадим несколько определений.

Левый (правый, двусторонний) идеал I в A называется *нильпотентным*, если для некоторого целого числа n выполняется $I^n = \{0\}$. Сумма всех нильпотентных левых идеалов алгебры A является нильпотентным двусторонним идеалом в A, называется *радикалом* A и обозначается $\operatorname{rad} A$. Известно (см. [5]), что $\operatorname{rad} A$ содержится в любом максимальном двустороннем идеале A. Так, например, если пересечение всех максимальных двусторонних идеалов A равно $\{0\}$, то $\operatorname{rad} A = \{0\}$. Обратно, если пересечение всех максимальных двусторонних идеалов A является левым нильпотентным идеалом, то оно совпадает с $\operatorname{rad} A$.

Элемент a алгебры A называется обратимым, если существует такой $a^{-1} \in A$, что $aa^{-1} = a^{-1}a = e$, где e — единица алгебры. Обозначим множество обратимых элементов алгебры A через $A^\times$. Алгебра D называется *алгеброй с делением*, если $D^\times = D \setminus \{0\}$. Алгебра A называется *локальной*, если фактор-алгебра $A/\operatorname{rad} A$ является алгеброй с делением, и A называется *основной*, если $A/\operatorname{rad} A$ является прямым произведением алгебр с делением. Будем называть алгебру A над полем k *сверхосновной* (см. [13]), если $A/\operatorname{rad} A \cong k^t$ для некоторого t. Очевидно, что любая сверхосновная алгебра является основной.

Теорема 2 (М. Блезер, см. [13]). *Алгебра A над полем k является алгеброй минимального ранга тогда и только тогда, когда*

$$A \cong C_1 \times \cdots \times C_s \times \underbrace{k^{2\times 2} \times \cdots \times k^{2\times 2}}_{u \text{ раз}} \times B, \tag{9}$$

где $C_1, \ldots, C_s$ суть локальные алгебры минимального ранга, у которых $\dim(C_\sigma/\operatorname{rad} C_\sigma) \geq 2$, то есть $C_\sigma \cong k[X]/(p_\sigma(X)^{d_\sigma})$ для некоторого неприводимого над k полинома p_σ, $\deg p_\sigma \geq 2$, $d_\sigma \geq 1$ и $\sharp k \geq 2\dim C_\sigma - 2$, $\sigma = 1, \ldots, s$, а B — сверхосновная алгебра минимального ранга над k. B является сверхосновной алгеброй минимального ранга тогда и только тогда, когда найдутся такие $w_1, \ldots, w_m \in \operatorname{rad} B$, $w_i w_j = 0$, $i \neq j$, что

$$\operatorname{rad} B = \mathsf{L}_B + Bw_1B + \cdots + Bw_mB = \mathsf{R}_B + Bw_1B + \cdots + Bw_mB$$

и $\sharp k \geq 2N(B)-2$, где L_B и R_B суть правый и левый аннигиляторы $\operatorname{rad} B$ (то есть множество всех таких $x \in \operatorname{rad} B$, что $x(\operatorname{rad} B) = 0$, соответственно $(\operatorname{rad} B)x = 0$), а $N(B)$ — наибольшее целое j, для которого $(\operatorname{rad} B)^j \neq \{0\}$. Любое из чисел s, и может равняться нулю, а множитель B является необязательным.

3 Умножение в групповых алгебрах

Билинейная и мультипликативная сложности умножения в групповых алгебрах изучались ранее в [2, 4, 3, 8, 9]. Приведём основные полученные результаты.

Теорема 3 (см. [9]). *Пусть k — алгебраически замкнутое поле простой характеристики p, G — абелева группа конечного порядка n, и $n = p^d t$, $p \nmid t$. Тогда существует такая сверхосновная алгебра B минимального ранга, что*

$$k[G] \cong B^t, \qquad \dim B = p^d.$$

При этом

$$\operatorname{rk} k[G] = C(k[G]) = 2n - t, \qquad \operatorname{rk} B = C(B) = 2p^d - 1.$$

Алгебра $k[G]$, согласно (9), является алгеброй минимального ранга.

Замечание. В действительности, условие теоремы можно ослабить, алгебраическая замкнутость поля k не является необходимым условием. Утверждение теоремы справедливо, если k лишь содержит ровно t различных корней степени t из единицы.

Теорема 4 (см. [9]). *Пусть k — алгебраически замкнутое поле характеристики 0, и G — абелева группа конечного порядка n. Тогда*

$$k[G] \cong k^n,$$

и

$$\operatorname{rk} k[G] = C(k[G]) = n.$$

Алгебра $k[G]$, согласно (9), является алгеброй минимального ранга.

Замечание. Снова алгебраическая замкнутость поля k не является необходимым условием. Утверждение теоремы будет также справедливо, если k лишь содержит ровно n различных корней степени n из единицы.

Теорема 5 (см. [9]). *Пусть G — абелева группа конечного порядка n, и*

$$G \cong \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_s}$$

(при этом $n = n_1 \cdots n_s$). Пусть m — число чётных чисел среди $n_1, \ldots, n_s$. Тогда

$$\mathbb{R}[G] \cong \mathbb{R}^{2^m} \times (\mathbb{R}[X]/(X^2+1))^{\frac{n-2^m}{2}},$$

и

$$\operatorname{rk}\mathbb{R}[G] = C(\mathbb{R}[G]) = \frac{3}{2}n - 2^{m-1}.$$

Алгебра $\mathbb{R}[G]$, согласно (9), является алгеброй минимального ранга.

Заметим, что все известные нижние оценки являются линейными относительно размерности рассматриваемых алгебр, а оптимальные алгоритмы обладают линейной сложностью. В этом случае представляет интерес асимптотика билинейной (а также мультипликативной) сложности, если её можно некоторым образом определить. Пусть A_1, A_2, ... — последовательность алгебр над полем k, n_1, n_2, ... — последовательность их размерностей а r_1, r_2, ... — последовательность их билинейных сложностей (натуральных чисел). Рассмотрим последовательность отношений

$$\frac{r_1}{n_1}, \frac{r_2}{n_2}, \ldots \tag{10}$$

(рациональных чисел). Если у этой последовательности существует предел, то назовём его *константой асимптотики сложности* для последовательности алгебр A_1, A_2, ... и обозначим его через c_A.

Заметим, что, согласно теоремам 3, 4 билинейная сложность в алгебрах над алгебраически замкнутыми полями полностью определяется их размерностью. В случае же поля вещественных чисел это неверно.

Теорема 6 (см. [8, 9]). *1. Пусть A_1, A_2, ... — коммутативные групповые алгебры над полем вещественных чисел. Если для последовательности A_1, A_2, ... существует константа асимптотики сложности c_A, то найдётся такое целое $i \geq 0$, что c_A равна одному из c_i, где*

$$c_0 = \frac{3}{2}, \qquad c_i = \frac{3}{2} - \frac{1}{2i},\ i \geq 1. \tag{11}$$

2. Для любого $i \geq 0$ существует последовательность коммутативных групповых алгебр над полем вещественных чисел A_1, A_2, ... с константой асимптотики сложности c_i (см. (11)).

3. Для любого $i \geq 1$ существует такая последовательность коммутативных групповых алгебр над полем вещественных чисел A_1, A_2, ..., что $\operatorname{rk} A_n = c_i \dim A_n$ для $n \geq 1$, то есть соответствующая последовательность (10) состоит из одних единиц.

4. Если последовательность коммутативных групповых алгебр над полем вещественных чисел A_1, A_2, ... имеет константу асимптотики сложности c_i, $i \geq 1$, то, начиная с некоторого $N \geq 1$ при $n \geq N$ выполняется $\operatorname{rk} A_n = c_i \dim A_n$.

5. Если у последовательности коммутативных групповых алгебр над полем вещественных чисел A_1, A_2, ... существует константа асимптотики сложности, которая равна c_0, то для любого $n \geq 1$ выполняется $\operatorname{rk} A_n > c_0 \dim A_n$.

Были изучены также некоторые некоммутативные алгебры, установлена их структура и сложность умножения.

Теорема 7 (см. [4, 3]). *Пусть S_3 — группа подстановок третьего порядка. Тогда*

$$\mathbb{C}[S_3] \cong \mathbb{C}^{2\times 2} \times \mathbb{C}^2, \qquad \mathbb{R}[S_3] \cong \mathbb{R}^{2\times 2} \times \mathbb{R}^2,$$
$$\operatorname{rk}\mathbb{C}[S_3] = C(\mathbb{C}[S_3]) = 9, \qquad \operatorname{rk}\mathbb{R}[S_3] = C(\mathbb{R}[S_3]) = 9.$$

В $\mathbb{C}[S_3]$ (в $\mathbb{R}[S_3]$) существует единственная подалгебра, изоморфная $\mathbb{C}^{2\times 2}$ (соответственно, изоморфная $\mathbb{R}^{2\times 2}$).

Теорема 8 (см. [2]). *Пусть Q — группа симметрий квадрата на плоскости (группа подстановок* (1)(2)(3)(4), (1234), (13)(24), (1432), (13), (24), (12)(34), (14)(23)*). Тогда*

$$\mathbb{C}[Q] \cong \mathbb{C}^{2\times 2} \times \mathbb{C}^4,$$

причём

$$\mathbb{C}[Q] \cong \mathbb{C}[H],$$

где H — группа, образованная гамильтоновыми кватернионами $1, -1, i, -i, j, -j, k, -k$ относительно умножения, и $Q \ncong H$.

4 Умножение полиномов многих переменных

Рассмотрим задачу умножения полиномов многих переменных и покажем, каким образом из быстрых алгоритмов умножения в коммутативных групповых алгебрах получаются эффективные алгоритмы умножения полиномов многих переменных.

Известно (см. [24]), что сложность умножения полиномов от m переменных степени c по каждой переменной не превосходит $c^m(2,5m + 2\log_2 c + 1)$ над полем, поддерживающим быстрое преобразование Фурье на c точках, и $O(c^m m \log c \log\log c)$ над произвольным полем. Мы покажем простой способ получения алгоритмов линейной мультипликативной сложности для умножения полиномов многих переменных.

Наш подход основан на том, что алгебру полиномов нескольких переменных можно вложить в коммутативную групповую алгебру с подходящим групповым базисом G. Разлагая G в прямое произведение циклических групп, достаточно обеспечить, чтобы для каждой переменной X_i, имеющей максимальную степень d_i в полиноме, в G был циклический сомножитель $\mathbb{Z}_{2d_i+1}$. Это обеспечит сохранение полной степени X_i в полиноме-произведении, которая, как легко видеть, может быть не более $2d_i$.

Теорема 9. *Пусть p_1 и p_2 — полиномы m переменных $X_1, \ldots, X_m$ степеней d_μ по переменным X_μ, $d_\mu > 0$, $1 \le \mu \le m$ над полем k, содержащем все корни степени $M = \prod_{\mu=1}^{m}(2d_\mu + 1)$ из 1. Пусть также p — характеристика поля k, и $M = p^d t$, где $p \nmid t$. Тогда существует билинейный алгоритм умножения полиномов p_1 и p_2 длины $2M - t$.*

Доказательство. Пусть $p_3 = p_1 \cdot p_2$. Очевидно, что p_3 имеет степень $2d_\mu$ по переменной X_μ, $1 \le \mu \le m$. Отсюда следует, что

$$p_3 = p_3 \bmod \prod_{\mu=1}^{m}(X_\mu^{2d_\mu+1} - 1) = p_3 \bmod (X_1^{2d_1+1} - 1) \cdots \bmod (X_m^{2d_m+1} - 1),$$

где mod означает остаток от деления левого операнда на правый[5]. Также очевидно, что

$$p_1 = p_1 \bmod (X_1^{2d_1+1} - 1) \cdots \bmod (X_m^{2d_m+1} - 1),$$
$$p_2 = p_2 \bmod (X_1^{2d_1+1} - 1) \cdots \bmod (X_m^{2d_m+1} - 1).$$

[5]Будем считать, что операция mod обладает левой ассоциативностью.

Рассмотрим алгебру $A = k[X_1, \ldots, X_m]/(X^{2d_1+1} - 1)/\cdots/(X^{2d_m+1} - 1)$ полиномов над k от $X_1, \ldots, X_m$ степеней d_μ по X_μ, $1 \le \mu \le m$. Порядок G, а также размерность $k[G]$, равны M. Алгебра A изоморфна алгебре $k[G]$ при

$$G = \mathbb{Z}_{2d_1+1} \times \cdots \times \mathbb{Z}_{2d_m+1}.$$

Действительно, обозначим элементы G через $g_{i_1,\ldots,i_m}$, $0 \le i_\mu \le 2d_m$, причём

$$g_{i_1,\ldots,i_m} \cdot g_{j_1,\ldots,j_m} = g_{(i_1+j_1) \bmod 2d_1, \ldots, (i_m+j_m) \bmod 2d_m}.$$

Тогда вложение

$$X_1^{i_1} X_2^{i_2} \cdots X_m^{i_m} \mapsto g_{i_1,i_2,\ldots,i_m},$$

как нетрудно проверить, определяет изоморфизм. Согласно теореме 3, $\operatorname{rk} B = 2M - t$, откуда следует утверждение теоремы.

Теорема доказана. □

Замечание. 1. Если k — поле характеристики 0, то существует билинейный алгоритм умножения полиномов p_1 и p_2 длины M.

2. Если $p \nmid \prod_{\mu=1}^{m}(2d_\mu + 1)$, то существует билинейный алгоритм умножения полиномов p_1 и p_2 длины M.

Константа M, определённая в теореме 9, как нетрудно видеть, является достижимым максимальным числом ненулевых коэффициентов полинома-произведения.

Заметим, что M делится на p тогда и только тогда, когда некоторые множители $(2d_i + 1)$ делятся на p. Для того, чтобы избавиться от таких множителей, мы можем осуществлять вложение в алгебру

$$k[\ldots, X_i, \ldots]/(X_1^{2d_1+1} - 1)/\ldots/ \bmod (X_i^{2d_i+2} - 1)/\ldots/(X_m^{2d_m+1} - 1),$$

повышая степень образа для всех таких переменных X_i. Для остальных множителей будем выбирать прежние многочлены для факторизации. В этом случае размерность алгебры-образа будет

$$M' = \prod_{\substack{\mu=1 \\ p \nmid 2d_\mu+1}}^{m} (2d_\mu + 1) \cdot \prod_{\substack{\mu=1 \\ p \mid 2d_\mu+1}}^{m} (2d_\mu + 2),$$

и вложение, согласно теореме 3, будет осуществляться в $k^{M'}$. В этом случае алгоритм умножения становится тривиальным и имеет сложность M'.

5 Умножение матриц 3×3

Рассмотрим алгебру чётных подстановок над полем k характеристики, отличной от 2. Базисом данной алгебры является группа A_4, состоящая из 12 подстановок:

$$\begin{array}{lll} e = g_1 = (1)(2)(3)(4), & g_2 = (12)(34), & g_3 = (13)(24), \\ g_4 = (14)(23), & g_5 = (1)(234), & g_6 = (2)(143), \\ g_7 = (3)(124), & g_8 = (4)(132), & g_9 = (1)(243), \\ g_{10} = (2)(134), & g_{11} = (3)(142), & g_{12} = (4)(123). \end{array}$$

Теорема 10. *Пусть характеристика k отлична от 2. Тогда в $k[A_4]$ существует подалгебра, изоморфная $k^{3\times 3}$.*

Доказательство. Пусть m_{ij}, $1 \le i, j \le 3$, обозначает матрицу порядка 3 над k, в которой в на пересечении i-й строки и j-го столбца стоит единица, а в остальных позициях расположены нули. Непосредственной проверкой можно установить, что вложение

$$
\begin{aligned}
&m_{11} \mapsto e_1 = \frac{1}{4}(g_1 + g_2 - g_3 - g_4), && m_{22} \mapsto e_2 = \frac{1}{4}(g_1 - g_2 + g_3 - g_4),\\
&m_{33} \mapsto e_3 = \frac{1}{4}(g_1 - g_2 - g_3 + g_4), && m_{12} \mapsto e_4 = \frac{1}{4}(g_5 + g_6 - g_7 - g_8),\\
&m_{23} \mapsto e_5 = \frac{1}{4}(g_5 - g_6 + g_7 - g_8), && m_{31} \mapsto e_6 = \frac{1}{4}(g_5 - g_6 - g_7 + g_8),\\
&m_{32} \mapsto e_7 = \frac{1}{4}(g_9 + g_{10} - g_{11} - g_{12}), && m_{13} \mapsto e_8 = \frac{1}{4}(g_9 - g_{10} + g_{11} - g_{12}),\\
&m_{21} \mapsto e_9 = \frac{1}{4}(g_9 - g_{10} - g_{11} + g_{12})
\end{aligned}
$$

является изоморфным.

Теорема доказана. □

Замечание. Заметим, что построенная подалгебра $k^{3\times 3}$ является линейным подпространством $k[A_4]$, определяемым уравнениями

$$
\begin{cases}
\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 0,\\
\alpha_5 + \alpha_6 + \alpha_7 + \alpha_9 = 0,\\
\alpha_9 + \alpha_{10} + \alpha_{11} + \alpha_{12} = 0,
\end{cases}
$$

где α_i — коэффициенты при g_i, $1 \le i \le 12$.

Теорема 11. *Имеет место изоморфизм*

$$\mathbb{C}[A_4] \cong \mathbb{C}^3 \times \mathbb{C}^{3\times 3}, \tag{12}$$

а также сложностное равенство

$$\operatorname{rk} \mathbb{C}[A_4] = \operatorname{rk} \mathbb{C}^{3\times 3} + 3. \tag{13}$$

Доказательство. Действительно, пусть ε — корень третьей степени из 1 в $\mathbb{C}$, отличный от 1. Тогда множество векторов $e_1, \ldots, e_{12}$, первые девять из которых определены выше в теореме 10, а

$$
\begin{aligned}
e_{10} &= \frac{1}{12}(g_1 + g_2 + g_3 + g_4 + g_5 + g_6 + g_7 + g_8 + g_9 + g_{10} + g_{11} + g_{12}),\\
e_{11} &= \frac{1}{12}(g_1 + g_2 + g_3 + g_4 + \varepsilon(g_5 + g_6 + g_7 + g_8) + \varepsilon^2(g_9 + g_{10} + g_{11} + g_{12})),\\
e_{12} &= \frac{1}{12}(g_1 + g_2 + g_3 + g_4 + \varepsilon^2(g_5 + g_6 + g_7 + g_8) + \varepsilon(g_9 + g_{10} + g_{11} + g_{12})),
\end{aligned}
$$

является базисом в $\mathbb{C}[A_4]$. При этом $e_i e_j = e_j e_i = 0$, $e_j^2 = e_j$, $1 \le i \le 9$, $10 \le j \le 12$; $e_r e_s = 0$, $10 \le r, s \le 12$, $r \ne s$.

Таким образом, каждый из вектором e_{10}, e_{11}, e_{12} порождает идеал, изоморфный $\mathbb{C}$. Отсюда, а также из теоремы 10, следует (12). Из гипотезы о прямой сумме[6] (см. [15]), которая является тривиально справедливой, когда один из сомножителей является степенью поля, следует (13).

Теорема доказана. □

[6] Гипотеза о прямой сумме гласит, что, если $C = A \times B$, то $\operatorname{rk} C = \operatorname{rk} A + \operatorname{rk} B$.

Теорема 12. *Имеет место изоморфизм*

$$\mathbb{R}[A_4] \cong \mathbb{R} \times \mathbb{R}[X]/(X^2+1) \times \mathbb{R}^{3\times 3}, \tag{14}$$

а также следующая нижняя оценка сложности умножения матриц третьего порядка

$$\operatorname{rk}\mathbb{R}^{3\times 3} \geq \operatorname{rk}\mathbb{R}[A_4] - 4. \tag{15}$$

Доказательство. множество векторов $e_1, \ldots, e_{12}$, первые десять из которых определены в теоремах 10, 11, а

$$e_{11} = \frac{1}{4}(2(g_1+g_2+g_3+g_4) - (g_5+g_6+g_7+g_8) - (g_9+g_{10}+g_{11}+g_{12})),$$
$$e_{12} = \frac{1}{4}(-(g_5+g_6+g_7+g_8) + (g_9+g_{10}+g_{11}+g_{12})),$$

является базисом в $\mathbb{R}[A_4]$. При этом $e_ie_j = e_je_i = 0$, $1 \leq i \leq 10$, $j = 11, 12$;

$$e_{11}e_{11} = e_{11}, \qquad e_{12}e_{12} = -e_{11},$$
$$e_{12}e_{11} = e_{12}, \qquad e_{11}e_{12} = e_{12}.$$

Таким образом, векторы e_{11}, e_{12} порождают идеал, изоморфный $\mathbb{R}[X]/(X^2+1)$. Вектор e_{10} порождает идеал, изоморфный $\mathbb{R}$. Отсюда, а также из теоремы 10, следует (14). Из (14) следует

$$\operatorname{rk}\mathbb{R}[A_4] \leq \operatorname{rk}\mathbb{R}^{3\times 3} + \operatorname{rk}\mathbb{R}[X]/(X^2+1) + 1 = \operatorname{rk}\mathbb{R}^{3\times 3} + 4,$$

так как $\operatorname{rk}\mathbb{R}[X]/(X^2+1) = 3$, откуда следует (13).

Теорема доказана. □

Заметим, что лучшая нижняя оценка (справедливая над произвольным полем) билинейной сложности умножения матриц третьего порядка принадлежит Блезеру (см. [14]) и равна 19. Таким образом, из (13) следует

$$\operatorname{rk}\mathbb{C}[A_4] \geq 22.$$

В то же время, известна верхняя оценка, получаемая из алгоритма Ладермана (см. [22]), и равная 23. Таким образом, из (13) и (15) следует

$$\operatorname{rk}\mathbb{C}[A_4] \leq 26, \qquad \operatorname{rk}\mathbb{R}[A_4] \leq 27.$$

Из теорем 11 и 12 можно получить следующую альтернативу.

Теорема 13. *Справедливо, по крайней мере, одно из следующих утверждений:*

1. $\operatorname{rk}\mathbb{C}^{3\times 3} < \operatorname{rk}\mathbb{R}^{3\times 3}$,
2. $\operatorname{rk}\mathbb{C}[A_4] < \operatorname{rk}\mathbb{R}[A_4]$,
3. *гипотеза о прямой сумме является неверной.*

Возникает вопрос о том, какая из альтернатив теоремы 13 является наиболее вероятной. С одной стороны, все имеющиеся нижние и верхние оценки ранга алгебры матриц порядка 3×3 совпадают для полей комплексных и вещественных чисел. Однако, разница между верхними и нижними оценками всё ещё относительно велика, что не даёт уверенности в утверждении, что сложность одинакова для вещественного и комплексного полей. С другой стороны, имеющиеся оценки сложности алгебры чётных подстановок четвёртого порядка различны. Однако, все они основаны на оценках сложности умножения матриц порядка 3×3, следовательно, последние зависят от этих оценок, и разница между верхними и нижними оценками также велика. Заметим, что гипотеза о прямой сумме неверна для приближённых алгоритмов (см., например, [1, 15]).

6 Заключение

В заключение сформулируем несколько открытых вопросов, связанных с установлением сложности умножения в групповых алгебрах и в алгебре матриц. Представляет интерес установление точного значения билинейной сложности умножения в коммутативных групповых алгебрах над произвольными полями, в частности, над полями, не содержащими нужный набор корней из единицы. Неизвестны хорошие верхние оценки и методы их получения над конечными полями, содержащими малое количество элементов. Представляет интерес получение нетривиальных оценок билинейной сложности последовательностей некоммутативных групповых алгебр растущей размерности, так как это приведёт к нетривиальным оценкам сложности умножения матриц.

Автор выражает благодарность своему научному руководителю профессору В. Б. Алексееву за введение в тему и многочисленные дискуссии. Автор также выражает благодарность профессорам М. Блезеру, М. Каминскому и А. Шпильке за полезные обсуждения и ценные замечания.

Работа выполнена при поддержке гранта РФФИ № 06–01–00438а.

Список литературы

[1] В. Б. Алексеев. *Сложность умножения матриц. Обзор.* Киберн. сб. (1988) **25**, 189–236.

[2] В. Б. Алексеев, А. Д. Поспелов. *Сложность умножения в групповой алгебре симметрий квадрата.* Тезисы 6-ой Международной конференции «Дискретные модели в теории управляющих систем» (2004), 8–11.

[3] В. Б. Алексеев, А. Д. Поспелов. *О ранге групповых алгебр.* «Математические методы решения инженерных задач», 5–25 (2005).

[4] В. Б. Алексеев, А. Д. Поспелов. *Сложность умножения в некоторых групповых алгебрах.* Дискретная математика (2005) **17**, №1, 3–17.

[5] Б. Л. Ван дер Варден. *Алгебра.* М.: Наука, 1979.

[6] Э. Б. Винберг. *Курс алгебры.* М., Изд-во «Факториал Пресс», 2001 г.

[7] А. А. Карацуба, Ю. П. Офман. *Умножение многозначных чисел на автоматах.* Докл. АН СССР, т. 145, № 2, с. 293—294 (1962).

[8] А. Д. Поспелов. *Ранг коммутативных групповых алгебр над полями комплексных и вещественных чисел.* Тезисы докладов XIV Международной конференции «Проблемы теоретической кибернетики» (Пенза, 23–28 мая 2005 г.), с. 125.

[9] А. Д. Поспелов. *Билинейная сложность умножения в коммутативных групповых алгебрах.* Сборник тезисов лучших дипломных работ 2005 года. Москва, издательский отдел ф-та ВМиК МГУ им. М. В. Ломоносова (2005), с. 75–76.

[10] А. Л. Тоом. *О сложности схемы из функциональных элементов, реализующей умножение целых чисел.* Доклады Академии Наук СССР, т. 150, № 2, с. 496–498 (1963).

[11] A. Alder, V. Strassen. *On the algorithmic complexity of associative algebras.* Theoret. Comput. Sci., 15:201–211, 1981.

[12] M. Bläser. *A $2.5n^2$-lower bound for the rank of $n \times n$-matrix multiplication over arbitrary fields.* Proc. 40th Ann. IEEE Symp. on Foundations of Comput. Sci. (FOCS), 45–50, 1999.

[13] M. Bläser. *Algebras of Minimal Rank over Arbitrary Fields.* SIIM Technical Report, 17 p., May 10, 2002.

[14] M. Bläser. *On the complexity of the multiplication of matrices of small formats.* J. Complexity 19(1): 43–60 (2003).

[15] P. Bürgisser, M. Clausen, M. Shokrollahi. *Algebraic Complexity Theory* Springer, 1997.

[16] P. Bürgisser, M. Lotz. *Lower Bounds on the Bounded Coefficient Complexity of Bilinear Maps.* Journal of the ACM 51(3): 464–482 (2004).

[17] H. Cohn, C. Umans. *A Group-Theoretic Approach to Fast Matrix Multiplication.* FOCS 2003: 438–449 (2003).

[18] H. Cohn, R. D. Kleinberg, B. Szegedy, C. Umans. *Group-theoretic Algorithms for Matrix Multiplication.* FOCS 2005: 379–388 (2005).

[19] D. Coppersmith and S. Winograd. *Matrix multiplication via arithmetic progressions.* Journal of Symbolic Computation, 9:251—280 (1990).

[20] M. Fürer. *Faster integer multiplication.* Proceedings of STOC 2007, 57–66 (2007).

[21] M. Kaminski. *A Lower Bound on the Complexity of Polynomial Multiplication over Finite Fields.* SIAM J. Comput. 34(4): 960–992 (2005).

[22] J. Laderman. *A noncommutative algorithm for multiplying* 3×3 *matrix using 23 multiplications.* Bull. Amer. Math. Soc. (1976) **82**, №1, 126–128

[23] V. Ya. Pan. *Strassen's Algorithm Is Not Optimal. Trilinear Technique of Aggregating, Uniting and Canceling for Constructing Fast Algorithms for Matrix Multiplication.* FOCS 1978: 166–176 (1978).

[24] V. Ya. Pan. *Simple multivariate polynomial multiplication.* Journal of Symbolic Computation, Volume 18, Issue 3, 183–186 (1994).

[25] R. Raz. *On the complexity of matrix product.* In Proceedings of the thirty-fourth annual ACM symposium on Theory of computing. ACM Press, 2002.

[26] A. Schönhage und V. Strassen. *Schnelle Multiplikation grosser Zahlen.* Computing, v. 7, pp. 281–292 (1971).

[27] A. Shpilka. *Lower bounds for matrix product.* In 42nd IEEE Symposium on Foundations of Computer Science, 2001.

[28] V. Strassen. *Gaussian elimination is not optimal.* Numer. Math. 13, 354–356 (1969).

УДК 681.322

О СТАБИЛИЗАЦИИ ЦЕПОЧКИ МНОЖЕСТВ ПРИ ПОСТРОЕНИИ МИНИМАЛЬНОЙ ВЫПУКЛОЙ ОБОЛОЧКИ

© 2008 г. А. И. Пучкова

apuchkova@gmail.com

Кафедра Оптимального управления

В данной работе изучается вопрос о стабилизации последовательности множеств специального вида. Эта последовательность используется для конструктивного описания минимальной выпуклой оболочки множества в E^n. Напомним ряд определений и утверждений, необходимых для формулировки и доказательства основного результата.

Определение 1. *Минимальной выпуклой оболочкой* $\mathrm{conv}F$ *множества* $F \subset E^n$ *называется наименьшее выпуклое множество, содержащее множество* F.

Определение 2. *Точка* x *называется выпуклой комбинацией точек* $x_1, \ldots, x_m \in E^n$, *если существуют числа* $\lambda_i,\ i = 1, \ldots, m$, *удовлетворяющие соотношениям* $\lambda_i \geq 0,\ \lambda_1 + \cdots + \lambda_m = 1$, *такие, что выполняется равенство*

$$x = \lambda_1 x_1 + \cdots + \lambda_m x_m.$$

Определение 3. *Пусть* x, y – *точки пространства* E^n. *Отрезком* $[x, y]$ *с концами* x, y *называется множество*

$$[x, y] = \{z \in E^n :\ z = \lambda x + (1 - \lambda)y,\ \lambda \in [0, 1]\},$$

или

$$[x, y] = \bigcup_{\lambda \in [0,1]} \{\lambda x + (1 - \lambda)y\}.$$

Существует два подхода к описанию минимальной выпуклой оболочки. Первый связан с именем Каратеодори. Он базируется на следующих двух утверждениях, доказательство которых можно найти в [2].

Лемма 1. *Для любого множества* $F \subset E^n$ *существует минимальная выпуклая оболочка* $\mathrm{conv}F$, *которая совпадает с совокупностью всех выпуклых комбинаций точек из множества* F.

Лемма 2 (теорема Каратеодори). *Пусть* $F \subset E^n$. *Любая точка* $x \in \mathrm{conv}F$ *представима в виде выпуклой комбинации не более чем* $n + 1$ *точек из множества* F, *где* n – *размерность пространства* E^n.

Второй подход к описанию минимальной выпуклой оболочки можно назвать геометрическим. В его основе лежит лемма 3, доказательство которой можно найти в [1].

Лемма 3 (о построении минимальной выпуклой оболочки). *Для любого множества* $F \subset E^n$ *существует минимальная выпуклая оболочка* $\mathrm{conv}F$, *которую можно построить следующим образом. Рассмотрим последовательность множеств*

$$F_0, F_1, \ldots, F_m, \ldots, \tag{1}$$

где

$$\begin{aligned} F_0 &= F, \\ F_1 &= \bigcup_{x,y\in F_0} [x,y], \\ F_2 &= \bigcup_{x,y\in F_1} [x,y], \\ \cdots\cdots\cdots & \quad \cdots\cdots\cdots \\ F_{m+1} &= \bigcup_{x,y\in F_m} [x,y], \\ \cdots\cdots\cdots & \quad \cdots\cdots\cdots \end{aligned}$$

Тогда $\mathrm{conv}F = \bigcup\limits_{m=0}^{\infty} F_m$.

Из определения цепочки множеств (1) видно, что построенная в лемме 3 последовательность множеств "разбухает" (обладает свойством монотонности по включению):

$$F_0 \subset F_1 \subset F_2 \subset \cdots \subset F_m \subset \bigcup_{m=0}^{\infty} F_m \equiv \mathrm{conv}F.$$

Заметим, что для выпуклого множества F имеем $F_0 = F_1 = \cdots = \mathrm{conv}F$, т. е. $F = \mathrm{conv}F$. Для множества, состоящего из двух точек в E^n, выпуклой оболочкой будет отрезок, соединяющий эти точки, $\mathrm{conv}F = F_1$.

Логично поставить вопрос: процесс построения минимальной выпуклой оболочки имеет бесконечный характер, или он заканчивается за конечное число шагов? Ответ на этот вопрос даёт следующая теорема – основной результат данной статьи.

Теорема (о стабилизации цепочки множеств (1)). *В конечномерном пространстве цепочка множеств* (1) *стабилизируется, т. е. существует номер* s*, зависящий от размерности пространства и множества* F $(s = s(n, F))$*, начиная с которого,*

$$F_s = F_{s+1} = \cdots = \mathrm{conv}F,$$

более того, для любого множества $F \subset E^n$ *имеет место следующая оценка сверху для* $s(n, F)$*:*

$$s(n, F) \le S(n) \equiv [\log_2 n] + 1, \tag{2}$$

где $[x]$ *– целая часть числа* x*.*

Доказательство. Сначала докажем вспомогательную лемму.

Лемма 4. *При* $k \ge 1$ *множество* F_k *в последовательности* (1) *состоит из выпуклых комбинаций не более чем* 2^k *точек исходного множества* F*.*

Доказательство. Используя метод математической индукции, имеем: при $k = 1$ утверждение леммы 4 верно, так как любая точка $z \in F_1$ представима в виде $z = \alpha x + (1 - \alpha)y$, где $x, y \in F_0 = F$, $0 \le \alpha \le 1$, поэтому эта точка является выпуклой комбинацией не более двух точек множества F.

База индукции состоит в предположении, что утверждение леммы 4 верно при каком-то $k > 1$. Осталось показать, что множество F_{k+1} состоит из выпуклых комбинаций не более чем 2^{k+1} точек из F. Пусть $z \in F_{k+1}$, тогда точку z можно представить в форме

$$z = \gamma x + (1 - \gamma)y, \quad x, y \in F_k,\ 0 \le \gamma \le 1.$$

Так как $x, y \in F_k$, то, применяя предположение базы индукции, получим, что

$$x = \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_{2^k} x_{2^k}, \quad y = \beta_1 y_1 + \beta_2 y_2 + \cdots + \beta_{2^k} y_{2^k},$$

где

$$x_1, \ldots, x_{2^k}, y_1, \ldots, y_{2^k} \in F; \quad \alpha_i \geq 0, \beta_i \geq 0,\ i = 1, \ldots, 2^k; \quad \sum_{i=1}^{2^k} \alpha_i = \sum_{i=1}^{2^k} \beta_i = 1.$$

Отсюда имеем

$$z = \gamma(\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_{2^k} x_{2^k}) + (1-\gamma)(\beta_1 y_1 + \beta_2 y_2 + \cdots + \beta_{2^k} y_{2^k}) =$$

$$= \gamma\alpha_1 x_1 + \gamma\alpha_2 x_2 + \cdots + \gamma\alpha_{2^k} x_{2^k} + (1-\gamma)\beta_1 y_1 + (1-\gamma)\beta_2 y_2 + \cdots + (1-\gamma)\beta_{2^k} y_{2^k},$$

и

$$\gamma\alpha_1 + \gamma\alpha_2 + \cdots + \gamma\alpha_{2^k} + (1-\gamma)\beta_1 + (1-\gamma)\beta_2 + \cdots + (1-\gamma)\beta_{2^k} =$$

$$= \gamma \sum_{i=1}^{2^k} \alpha_i + (1-\gamma) \sum_{i=1}^{2^k} \beta_i = \gamma + (1-\gamma) = 1.$$

Таким образом, точка z является выпуклой комбинацией точек $x_1, \ldots, x_{2^k}, y_1, \ldots, y_{2^k}$ множества F. Очевидно, что различных точек здесь не более, чем 2^{k+1}. Лемма 4 доказана.

Используя результат леммы 4, можно утверждать, что при $s \geq 1$ множество F_s в цепочке (1) – это совокупность выпуклых комбинаций не более чем 2^s точек исходного множества F. С другой стороны, по теореме Каратеодори $\mathrm{conv}F$ является совокупностью выпуклых комбинаций не более чем $n+1$ точек множества F. Следовательно, выбирая s из условия $2^s > n$, получаем, что $F_s = \mathrm{conv}F$, т. е. последовательность множеств (1) стабилизируется на s-ом шаге. Условие $2^s > n$ влечёт неравенство $s > \log_2 n$, откуда видно, что s достаточно положить равным целому числу

$$S(n) = [\log_2 n] + 1 > \log_2 n.$$

Теорема доказана.

Для наглядности приведём таблицу значений $S(n)$ для некоторых n.

n	1	2	3	4	5	6	7	8	9	15	16	17	32	128	200
S(n)	1	2	2	3	3	3	3	4	4	4	5	5	6	8	8

В заключение отметим, что оценка (2) является точной. Для полноразмерного симплекса её нельзя улучшить, т. е. стабилизация наступает в точности на шаге с номером $S(n)$.

Так, например, для множества F, состоящего из трёх различных точек на плоскости, не лежащих на одной прямой, имеем: $n = 2$, $s(2, F) = S(2) = 2$. Для множества $F = \{f_1, f_2, \ldots, f_6, f_7\} \subset E^6$, состоящего из семи точек $f_1 = (1,1,1,1,1,1)$, $f_2 = (1,0,0,0,0,0)$, $f_3 = (0,1,0,0,0,0), \ldots, f_7 = (0,0,0,0,0,1)$, имеем: $s(6, F) = S(6) = 3$.

Автор выражает признательность своему научному руководителю М.В. Орлову и Ю.Н. Киселёву за ряд ценных советов и замечаний.

Список литературы

[1] Киселёв Ю. Н. *Оптимальное управление.* Изд-во МГУ, 1988.

[2] Благодатских В. И. *Введение в оптимальное управление.* М.: Высшая школа, 2001.

УДК 681.322

ОБ АБСОЛЮТНОЙ ПОСТОЯННОЙ В НЕРАВЕНСТВЕ БЕРРИ–ЭССЕЕНА

© 2008 г. И. Г. Шевцова

ishevtsova@cs.msu.su

Кафедра Математической статистики

1 Постановка задачи и её история

Пусть $X_1, \ldots, X_n$ — независимые одинаково распределённые случайные величины, удовлетворяющие условиям

$$\mathsf{E}X_1 = 0, \quad \mathsf{D}X_1 = 1. \tag{1}$$

Согласно центральной предельной теореме теории вероятностей, в этом случае имеет место равномерная сходимость последовательности функций распределения центрированной и нормированной суммы случайных слагаемых к функции распределения стандартного нормального закона:

$$\rho(F_n, \Phi) \equiv \sup_x |F_n(x) - \Phi(x)| \longrightarrow 0, \quad n \to \infty, \tag{2}$$

где

$$F_n(x) = \mathsf{P}\left(\frac{X_1 + \ldots + X_n}{\sqrt{n}} < x\right), \quad \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt.$$

Указанный факт широко используется при решении задач, возникающих, например, в теории управления запасами, в страховой и финансовой математике, теории измерений, теории надежности, методах Монте–Карло и многих других важных областях. Однако на практике объем выборки всегда конечен и для эффективного применения справедливой при больших n аппроксимации $F_n(x) \approx \Phi(x)$ необходимо иметь разумные оценки скорости сходимости в (2). Условий (1) для этого оказывается недостаточно. Действительно, согласно результату Мацкявичуса [5], как бы медленно ни сходилась к нулю последовательность неотрицательных чисел δ_n, найдется последовательность независимых одинаково распределенных центрированных случайных величин с конечными вторыми моментами, такая что для функции распределения F_n их нормированной суммы выполняется неравенство

$$\rho(F_n, \Phi) \geqslant \delta_n,$$

справедливое при всех n, начиная с некоторого.

Классическим дополнительным условием, позволяющем строить стремящиеся к нулю с ростом числа слагаемых оценки, является ограниченность третьего абсолютного момента случайных слагаемых:

$$\beta \equiv \mathsf{E}|X_1|^3 < \infty. \tag{3}$$

Для этого случая в 1941-42гг. Э. Берри [14] и К.-Г. Эссеен [15] практически одновременно и независимо друг от друга доказали следующее неравенство, которое сейчас носит имена обоих своих авторов:

$$\rho(F_n, \Phi) \leqslant C \cdot \varepsilon, \quad \text{где} \quad \varepsilon = \frac{\beta}{\sqrt{n}}, \tag{4}$$

а C — положительная абсолютная постоянная. История отыскания наименьшего возможного значения этой постоянной

$$C_0 = \inf\{C\colon \rho(F_n, \Phi) \leqslant C \cdot \varepsilon \quad \text{для всех } n \geqslant 1\}$$

представляет собой отдельную историю, не менее увлекательную, чем сама задача. Так, Э. Берри [14] утверждал, что $C_0 \leqslant 1.88$, однако, как обнаружил позднее П. Л. Сюй [17], вычисления Берри содержали ошибку. К.-Г. Эссеен [15] показал, что $C_0 \leqslant 7.59$. Х. Бергстрём [13] снизил оценку до $C_0 \leqslant 4.8$. К. Такано [20] получил результат $C_0 \leqslant 2.031$. По-видимому, работа Такано (опубликованная на японском языке) выпала из поля зрения некоторых исследователей, так как в нескольких более поздних публикациях приводятся немного худшие оценки. В частности, в работе Эссеена [16] имеется упоминание о неопубликованных вычислениях, приводящих к неравенству $C_0 \leqslant 2.9$. Д. Л. Уоллес [22] получил оценку $C_0 \leqslant 2.05$. В. Феллер [8], упоминая результат Уоллеса, также обходит вниманием работу Такано. Вычислению точного значения C_0 придавал большое значение А. Н. Колмогоров. В работе [2] он высказал предположение о том, что $C_0 = 1/\sqrt{2\pi}$. К сожалению, это предположение оказалось не совсем точным: в 1956 г., решая несколько иную задачу, К.-Г. Эссеен [16] показал, что в (4) постоянная C_0 не может быть меньше, чем

$$C_1 = \frac{\sqrt{10}+3}{6\sqrt{2\pi}} = \frac{1}{\sqrt{2\pi}} + 0.0107899...$$

Этот результат получен как следствие решения задачи о наименьшей постоянной C_*, обеспечивающей асимптотическую оценку

$$\rho(F_n, \Phi) \leqslant C_* L_n^3 + o(L_n^3).$$

Эссеен показал, что в рассматриваемой ситуации $C_* = C_1$. Поскольку $C_0 \geqslant C_*$, была найдена нижняя оценка для C_0. Однако, как позже установил Б. А. Рогозин [7],

$$\limsup_{n\to\infty} \inf_{a,b} \frac{\sigma^3\sqrt{n}}{\beta} \sup_x \left| F_n(x) - \Phi\Big(\frac{x-a}{b}\Big)\right| \leqslant \frac{1}{\sqrt{2\pi}} \leqslant 0.3990.$$

Тем самым предположение Колмогорова было в определенном смысле подтверждено.

Тем не менее, наименьшее возможное значение C_0 абсолютной постоянной в классическом неравенстве Берри–Эссеена до сих пор неизвестно. В 1966-67 гг. В. М. Золотарёв показал, что $C_0 \leqslant 0.9051$ [1] и $C_0 \leqslant 0.8197$ [23]. Эти результаты основаны на неравенстве сглаживания с адаптивным ядром, являющимся некоторой плотностью распределения вероятностей. При адаптации ядра существенную роль играла численная оптимизация, осуществленная на ЭВМ того времени. В 1971-72 гг. П. Ван Беек [11, 12] усовершенствовал метод Золотарева, и, используя знакопеременные сглаживающие ядра, получил оценку $C_0 \leqslant 0.7975$. Следующий рекорд был установлен И. С. Шигановым [10] десять лет спустя: $C_0 \leqslant 0.7655$. При этом в качестве адаптивного ядра использовалась функция более общего вида, с бóльшим количеством параметров. Следует отметить, что указанное ядро фактически было предложено еще Ван Бееком [12], но соответствующие вычисления не были им реализованы, по всей видимости, в силу недостаточной производительности доступных тогда ЭВМ.

Тем временем в 1972 г. был опубликован весьма интересный результат Г. Правитца [19]:

$$C_0 \leqslant \begin{cases} 0.5151, & \varepsilon' \leqslant 0.1, \\ 0.5270, & \varepsilon' \leqslant 1/7, \end{cases} \quad \text{где } \varepsilon' = \left(\frac{n}{n-1}\right)^{3/2} \cdot \varepsilon.$$

из которого вытекают следующие “условные” верхние оценки абсолютной постоянной в неравенстве Берри–Эссеена:

$$C_0 \leqslant \begin{cases} 0.5151, & \varepsilon \leqslant 0.0985, \\ 0.5270, & \varepsilon \leqslant 0.1387, \end{cases}$$

с учетом которых методом Шиганова можно было бы получить несколько более точную безусловную оценку постоянной C_0. Однако результат Правитца по какой-то причине выпал из поля зрения Шиганова. Лишь недавно было замечено, что с помощью результата Правитца можно уточнить оценку Шиганова, в результате чего появился нынешний “мировой рекорд”: $C_0 \leqslant 0.7056$ [9].

Среди работ, посвященных "условным" верхним оценкам абсолютной постоянной в неравенстве Берри–Эссеена, также следует упомянуть работу С. В. Нагаева и В. И. Чеботарева [6], в которых приведен следующий результат:

$$C_0 \leqslant 0.515489 + R, \quad \text{где} \quad R < \frac{1}{\beta_3}\left(0.427675 + \frac{0.153597}{\sqrt{n}}\right).$$

В нижеследующей таблице приведено несколько вытекающих отсюда "условных" верхних оценок C_0:

	$n \geqslant 3$	$n \geqslant 10$
$\beta_3 \geqslant 3$	$C_0 \leqslant 0.687607$	$C_0 \leqslant 0.674238$
$\beta_3 \geqslant 4$	$C_0 \leqslant 0.644578$	$C_0 \leqslant 0.634551$
$\beta_3 \geqslant 10$	$C_0 \leqslant 0.567124$	$C_0 \leqslant 0.563114$

2 Основной результат

Теорема 1. *В предположениях* (1) *и* (3) *справедлива оценка*

$$C_0 \leqslant 0.7005.$$

Доказательство. В основе доказательства лежит неравенство сглаживания Золотарёва [1, 23], улучшенное Ван Бееком [11, 12], а также вышеупомянутый результат Г. Правитца [19]. Ниже мы опишем лишь ключевые этапы, формулируя соответствующие утверждения в виде лемм.

Пусть $p(x) \in L_1$ – некоторая интегрируемая функция. Обозначим

$$\hat{p}(t) = \int e^{itx} p(x)dx, \quad \|p\| = \int |p(x)|dx, \quad p^+(x) = \max\{p(x), 0\}.$$

Для произвольных $x, y > 0$ введём функции

$$V(x) = x\int_0^x p^+(u)du, \quad q(y) = \frac{1}{\sqrt{2\pi}}\int_0^\infty |\hat{p}(t/y)|\frac{\delta(t)}{t}dt, \quad \delta(t) = |f_n(t) - e^{-t^2/2}|,$$

где $f_n(t)$ – характеристическая функция, соответствующая функции распределения $F_n(x)$. Обозначим через Λ класс всех непрерывных симметричных функций $p \in L_1$, таких что $\hat{p} \in L_1$.

Лемма 1 (см. [11]). *Для любой функции $p \in \Lambda$ такой, что уравнение*

$$4V(x) - x\,\|p\| = 0 \tag{5}$$

имеет положительный корень χ_p, при всех $x > \chi_p$ и $y > 0$ справедливо неравенство

$$\rho(F_n, \Phi) \leqslant \sqrt{\frac{2}{\pi}} \cdot \frac{V(x)/y + q(y)}{4V(x) - x\,\|p\|}.$$

Замечание 1. Поскольку правая часть последнего неравенства инвариантна относительного умножения ядра p на положительное число, можно ограничиться классом ядер $p \in \Lambda$, удовлетворяющих условию нормировки $\|p\| = 1$.

Следующая лемма дает оценку $\delta(t)$ (влияющую на значения величины $q(y)$), равномерную в классе всех распределений, удовлетворяющих условиям (1) и (3).

Лемма 2 ([1]). 1°. *Для* $|t| < \sqrt{2n}$

$$\delta(t) \leqslant \delta_1(t) = e^{-t^2/2}\left(\exp\left\{t^2\tau\left(|t|n^{-1/2},\beta\right)\right\} - 1\right), \quad \tau(u,\beta) = \frac{\beta u}{6} - \frac{1}{u^2}\left[\ln\left(1 - \frac{u^2}{2}\right) + \frac{u^2}{2}\right].$$

2°. *Для всех* $t \in \mathbb{R}$

$$\delta(t) \leqslant \delta_2(t) = e^{-t^2/2}(\exp\{k\varepsilon|t|^3/2\} + 1), \quad k = 4\sup_{x>0}\{(\cos x - 1 + x^2/2)/x^3\} \approx 0.396648.$$

3°. *Для всех* $t \in \mathbb{R}$

$$\delta(t) \leqslant \delta_3(t) = 1 + e^{-t^2/2}.$$

Доказательство. Последняя оценка 3° очевидна. Второе утверждение доказано Правитцом в работе [18] (см. также книгу Н. Г. Ушакова [21]). Утверждение 1° сформулировано Золотаревым в [1], но мы приведем здесь полное доказательство. Обозначим через $f(t)$ характеристическую функцию с.в. X_1: $f(t) = \mathsf{E}e^{itX_1}$, $t \in \mathbb{R}$. Тогда

$$f_n(t) = \left(f\left(\frac{t}{\sqrt{n}}\right)\right)^n.$$

Из разложения $f(t)$ в ряд Маклорена вытекают оценки

$$|f(t) - 1| \leqslant \frac{t^2}{2}, \quad |t| \leqslant \sqrt{2}, \quad \text{и} \quad f(t) = 1 - \frac{t^2}{2} + \theta_1\frac{\beta_3|t|^3}{6}, \quad t \in \mathbb{R}, \tag{6}$$

с некоторым $\theta_1 \in \mathbb{C}$, $|\theta_1| \leqslant 1$. Воспользуемся следующим справедливым для любых $r \in \mathbb{C}$, $|r| \leqslant 1$, разложением логарифма:

$$\ln(1+r) = r + \theta_2(\ln(1-|r|) + |r|), \quad \text{где } \theta_2 \in \mathbb{C},\ |\theta_2| \leqslant 1,$$

с $r = (f(t/\sqrt{n}) - 1)$ (в силу (6) при $|t| \leqslant \sqrt{2n}$ справедлива оценка $|r| \leqslant t^2/(2n) \leqslant 1$). Имеем

$$\ln(1+r) = \ln f\left(\frac{t}{\sqrt{n}}\right) = -\frac{t^2}{2} + \theta_1\frac{\beta_3|t|^3}{6n^{3/2}} + \theta_2(\ln(1-|r|) + |r|) =$$

$$= -\frac{t^2}{2n} + \theta_3\left(\frac{\beta_3|t|^3}{6n^{3/2}} - \ln\left(1 - \frac{t^2}{2n}\right) - \frac{t^2}{2n}\right) \equiv -\frac{t^2}{2n} + \theta_3 t^2\tau\left(\frac{|t|}{\sqrt{n}},\beta\right),$$

с некоторым $\theta_3 \in \mathbb{C}$, причем $|\theta_3| \leqslant 1$. Из последнего соотношения с учетом неравенства $|e^z - 1| \leqslant e^{|z|} - 1$, справедливого при всех $z \in \mathbb{C}$, получаем

$$\left|f_n(t) - e^{-t^2/2}\right| = e^{-t^2/2}\left|\exp\left\{\theta_3 t^2\tau\left(\frac{|t|}{\sqrt{n}},\beta\right)\right\} - 1\right| \leqslant e^{-t^2/2}\left(\exp\left\{t^2\tau\left(\frac{|t|}{\sqrt{n}},\beta\right)\right\} - 1\right),$$

что и требовалось доказать. □

Положим

$$\delta_0(t,\varepsilon,n) = \min\{\delta_1(t),\delta_2(t),\delta_3(t)\}, \quad q(y,\varepsilon,n) = \frac{1}{\sqrt{2\pi}}\int_0^\infty |\hat{p}(t/y)|\frac{\delta_0(t,\varepsilon,n)}{t}\,dt.$$

Вышеприведенные леммы позволяют искать оценку константы C_0 в виде

$$C_0 \leqslant C \equiv \sup_\varepsilon D(\varepsilon), \quad D(\varepsilon) \equiv \sup_n D(\varepsilon,n), \quad D(\varepsilon,n) \equiv \inf\{D(\varepsilon,n,x,y,p)\colon x > \chi_p,\ y > 0,\ p \in \Lambda\},$$

где

$$D(\varepsilon,n,x,y,p) \equiv \sqrt{\frac{2}{\pi}} \cdot \frac{V(x)/y + q(y,\varepsilon,n)}{\varepsilon(4V(x) - x\,\|p\|)}.$$

Следующие две леммы полезны при вычислении последних двух супремумов: $\sup_\varepsilon D(\varepsilon)$ и $\sup_n D(\varepsilon,n)$ при каждом фиксированном ε . При этом первый оценивается по значениям $D(\varepsilon)$ в конечном числе точек, а второй вычисляется в явном виде.

Лемма 3 ([1, 10]). *Для любых* $0 < \varepsilon_1 \leqslant \varepsilon \leqslant \varepsilon_2$ *справедливо соотношение*

$$D(\varepsilon) \leqslant D(\varepsilon_2) \cdot \frac{\varepsilon_2}{\varepsilon_1}.$$

Лемма 4. *При каждом фиксированном* $\varepsilon > 0$ *функция* $D(\varepsilon, n)$ *монотонно убывает по* n.

Доказательство. Заметим, что $D(\varepsilon, n, x, y, p)$ зависит от n только через функцию $\delta_0(t)$, причем эта зависимость монотонна. Из представления

$$\tau\left(\frac{|t|}{\sqrt{n}}, \beta\right) = \frac{\varepsilon|t|}{6} + \frac{t^2}{4n}\sum_{r=2}^{\infty}\frac{1}{r}\left(\frac{t^2}{2n}\right)^{r-2},$$

очевидно, вытекает, что $\tau\left(|t|n^{-1/2}, \beta\right)$, а значит, и $\delta_1(t)$ монотонно убывают по n при фиксированных ε и t. Поскольку с ростом n множество, по которому берется минимум

$$\delta_0(t) = \begin{cases} \min\{\delta_1(t), \delta_2(t), \delta_3(t)\}, & |t| < \sqrt{2n}, \\ \min\{\delta_2(t), \delta_3(t)\}, & |t| \geqslant \sqrt{2n}, \end{cases}$$

расширяется, величина $\delta_0(t)$, а значит, и $D(\varepsilon, n, x, y, p)$ монотонно убывает с ростом n. Итак, для любых $n_1 \leqslant n_2$ имеем $D(\varepsilon, n_2, x, y, p) \leqslant D(\varepsilon, n_1, x, y, p)$ при фиксированных ε, x, y и p. Отсюда вытекает, что

$$D(\varepsilon, n_2) \equiv \inf_{x,y,p} D(\varepsilon, n_2, x, y, p) \leqslant \inf_{x,y,p} D(\varepsilon, n_1, x, y, p) \equiv D(\varepsilon, n_1),$$

что и требовалось доказать. □

Из последней леммы вытекает, что супремум по n функции $D(\varepsilon, n)$ достигается при минимальном $n \geqslant 1$, допустимом при данном значении ε. Можно убедиться, что такое значение равняется $\lceil \varepsilon^{-2} \rceil$, где $\lceil x \rceil$ — минимальное целое, превосходящее или равное x. Действительно, из неравенства Ляпунова с учетом моментных условий (1) вытекает, что $\beta_3 \geqslant 1$, откуда получаем, что $n = \beta_3/\varepsilon^2 \geqslant 1/\varepsilon^2$ и

$$D(\varepsilon) \equiv \sup_n D(\varepsilon, n) = D\big(\varepsilon, \max\{1, \lceil \varepsilon^{-2} \rceil\}\big).$$

Укажем теперь область значений ε, по которой берется последний супремум $\sup_\varepsilon D(\varepsilon)$. Из результата Правитца [19] вытекает справедливость утверждения теоремы при $\varepsilon \leqslant 0.1387$. Кроме того, поскольку равномерное расстояние между двумя функциями распределения не превосходит единицы, достаточно рассматривать $\varepsilon \leqslant 1/C < 1.427552$, где $C = 0.7005$ — значение, объявленное в формулировке теоремы. Итак, окончательно получаем

$$C = \sup\{D(\varepsilon) \colon 0.1387 < \varepsilon < 1.427552\} = \sup\{D\big(\varepsilon, \max\{1, \lceil \varepsilon^{-2} \rceil\}\big) \colon 0.1387 < \varepsilon < 1.427552\}.$$

Нахождение внутреннего инфимума функционала $D(\varepsilon, n, x, y, p)$ сведем сначала к более простой задаче путем параметризации класса ядер. Будем рассматривать ядра вида

$$p(x) = \frac{1}{2}\sum_{k=1}^{6} w_k(p_1(x + a_k) + p_1(x - a_k)), \quad p_1(x) = \frac{\sin x}{2\pi x(1 - x^2/\pi^2)},$$

где a_k, w_k — произвольные действительные числа, $k = \overline{1,6}$, такие что $\chi_p < \infty$. Преобразование Фурье указанного ядра имеет вид

$$\hat{p}(t) = \hat{p}_1(t)\sum_{k=1}^{6} w_k \cos(a_k t), \quad \hat{p}_1(t) = \cos^2(\pi t/2)\mathbf{1}(|t| \leqslant 1),$$

где $\mathbf{1}(\cdot)$ — индикаторная функция. Таким образом, задача минимизации функционала $D(\varepsilon, n, x, y, p)$ свелась к задаче минимизации функции $D(\varepsilon, n, x, y, w_1, \ldots, w_6, a_1, \ldots, a_6)$ двенадцати переменных при фиксированных ε и n (на самом деле, свободных переменных всего одиннадцать, поскольку неравенство сглаживания Золотарева инвариантно относительно умножения ядра на положительное число, и поэтому один из весов можно зафиксировать).

Численная оптимизация проводилась по методу сопряжённых градиентов на компьютере с процессором Intel Core 2Duo T5500 1.6 ГГц. При написании соответствующей программы использовалась библиотека математических функций GNU Scientific Library (GSL) на языке C версии 1.8, домашняя страница http://www.gnu.org/software/gsl/. В ходе оптимизации оказалось, что экстремальные значения функции $D(\varepsilon)$ появляются при $\varepsilon \approx 0.5$, и они не превосходят 0.7005. Результаты соответствующих вычислений приведены в нижеследующей таблице.

ε	$D(\varepsilon)$	ε	$D(\varepsilon)$	ε	$D(\varepsilon)$	ε	$D(\varepsilon)$
1.427562	0.468717	0.500446	0.700222	0.392423	0.689863	0.203257	0.682088
0.955208	0.571401	0.500247	0.700285	0.386464	0.691067	0.197915	0.683072
0.779167	0.643003	0.500094	0.700334	0.381260	0.692151	0.192991	0.684235
0.715213	0.671486	0.499975	0.685148	0.376716	0.685094	0.188510	0.684272
0.685590	0.653474	0.489017	0.688220	0.368431	0.686651	0.184143	0.685347
0.639564	0.672034	0.480445	0.690581	0.361147	0.688254	0.180159	0.685928
0.613575	0.682321	0.473641	0.692468	0.354834	0.689579	0.176412	0.686004
0.597651	0.688549	0.468211	0.693882	0.349302	0.683820	0.172761	0.687048
0.587455	0.692415	0.463787	0.695034	0.340984	0.685472	0.169443	0.687886
0.580674	0.694927	0.460168	0.695979	0.333669	0.687335	0.166392	0.688005
0.576055	0.675834	0.457198	0.696758	0.327398	0.682713	0.163424	0.688878
0.555771	0.682696	0.454756	0.697405	0.319085	0.684106	0.160713	0.689756
0.541645	0.687401	0.452747	0.697921	0.311617	0.680886	0.158248	0.690252
0.531516	0.690667	0.451080	0.698343	0.302892	0.683188	0.155933	0.690337
0.524055	0.693006	0.449691	0.698705	0.295406	0.681344	0.153671	0.691041
0.518448	0.694633	0.448539	0.699005	0.287328	0.680643	0.151595	0.691634
0.514106	0.695968	0.447581	0.699244	0.279183	0.684061	0.149677	0.692185
0.510780	0.697004	0.446779	0.687580	0.272632	0.680872	0.147900	0.692685
0.508231	0.697811	0.438538	0.689511	0.264993	0.678954	0.146250	0.693128
0.506280	0.698430	0.431659	0.691071	0.256842	0.679447	0.144711	0.693515
0.504784	0.698878	0.425849	0.692397	0.249123	0.679442	0.143268	0.693921
0.503615	0.699247	0.420923	0.693495	0.241634	0.678818	0.141923	0.694251
0.502714	0.699529	0.416714	0.694366	0.234155	0.679434	0.140657	0.694589
0.502017	0.699731	0.413065	0.695138	0.227113	0.680124	0.139470	0.694898
0.501466	0.699899	0.409903	0.695853	0.220507	0.680895		
0.501036	0.700035	0.407184	0.686845	0.214335	0.681669		
0.500703	0.700140	0.399247	0.688527	0.208573	0.682646		

3 Методы дальнейшего уточнения абсолютной постоянной в неравенстве Берри-Эссеена

Изучая зависимость коэффициента C_ε при ляпуновской дроби в неравенстве Берри–Эссеена от ε,

$$C_\varepsilon = \inf\left\{D(\varepsilon)\colon \rho(F_n, \Phi) \leqslant D(\varepsilon)\cdot\varepsilon \quad \text{для всех } n \geqslant \max\left\{1, \varepsilon^{-2}\right\}\right\},$$

В. М. Золотарев высказал предположение, что асимптотическое значение этого коэффициента при $\varepsilon \to 0$ совпадает с C_0:

$$C_0 = \lim_{\varepsilon\to 0} C_\varepsilon.$$

В нашей же работе областью экстремальных значений C_ε оказались значения аргумента ε, отделенные от нуля. С одной стороны, указанная гипотеза позволяет надеяться, что задача нахождения оптимального ядра с целью уточнения C_ε для ε, отделенных от нуля, имеет смысл, и задачу эту можно решать, как минимум до тех пор, пока не удастся получить константу Правитца. С другой стороны, гипотеза Золотарева пробуждает интерес к исследованию асимптотического поведения C_ε при малых ε.

Теоретическое решение первой задачи было предложено еще Ван Бееком [12], доказавшим следующий результат. Обозначим

$$D^*(\varepsilon,n,p)=\inf\{D(\varepsilon,n,x,y,p)\colon x>\chi_p,\ y>0\}=\inf_{\substack{x>\chi_p,\\ y>0}}\left\{\sqrt{\frac{2}{\pi}}\cdot\frac{V(x)/y+q(y,\varepsilon,n)}{\varepsilon(4V(x)-x\,\|p\|)}\right\}.$$

Теорема 2. *Для любых фиксированных $n\geqslant 1$ и $\varepsilon>0$*

$$\inf_{p\in\Lambda}D^*(\varepsilon,n,p)=\inf_{p\in\Lambda_0}D^*(\varepsilon,n,p)=\inf_{p\in\Lambda_1}D^*(\varepsilon,n,p),$$

где Λ — класс непрерывных симметричных функций $p\in L_1$, таких что $\hat p\in L_1$, Λ_1 — класс функций $p\in\Lambda$ таких, что $\hat p$ имеет компактный носитель, то есть множество $\operatorname{supp}\hat p=\{t\colon \hat p(t)>0\}$ *является замкнутым и ограниченным, Λ_0 — класс функций $p\in\Lambda_1$ для которых отрезок $[-1,\,1]$ является минимальным отрезком таким, что* $\operatorname{supp}\hat p\subseteq[-1,\,1]$.

Пусть $h(t)\in\Lambda_1$ — произвольная функция, такая что $\hat h(t)$ не обращается в нуль на отрезке $[-1,\,1]$. Обозначим

$$p_m(x,\mathbf{a},\mathbf{w})=\frac{1}{2}\sum_{i=1}^{m}w_i(h(x+a_i)+h(x-a_i)),\tag{7}$$

где $\mathbf{a}=(a_1,\ldots,a_m)$, $\mathbf{w}=(w_1,\ldots,w_m)$, $a_i,\ w_i\in\mathbb{R}$, $i=\overline{1,m}$. Заметим, что

$$\hat p_m(t,\mathbf{a},\mathbf{w})=\hat h(t)\sum_{i=1}^{m}w_i\cos(a_it).$$

Теорема 3. *Для определенной выше последовательности функций $\{p_m\}_{m\geqslant1}$ имеем*

$$\inf_{p\in\Lambda}D^*(\varepsilon,n,p)=\lim_{m\to\infty}\inf_{\mathbf{a},\mathbf{w}\in\mathbb{R}^m}D^*(\varepsilon,n,p_m).$$

Доказательство теорем 2 и 3 можно найти в работе Ван Беека [12].

Таким образом, минимизацию функционала $D(\varepsilon,n,p)$ по классу всех ядер $p\in\Lambda$ можно заменить минимизацией функции $D(\varepsilon,n,p_m)$ по переменным $a_i,\ w_i\in\mathbb{R}$, $i=\overline{1,m}$ с достаточно большим m. Однако, слишком большие значения m снижают эффективность работы численных методов многомерной оптимизации, поэтому необходим некоторый компромисс: нужно использовать максимальное m, при котором эффективность работы численных методов достаточно велика. Так, например, для некоторых порождающих ядер $h(t)\in\Lambda_1$ автором была проведена численная оптимизация $D(\varepsilon,n,p_m)$ с p_m, определенным в (7). Однако текущая реализация программы, производящей необходимые вычисления, позволила провести расчеты лишь для m, не превосходящих $5-6$. Ниже приведены соответствующие результаты.

$h(x)$	m	C	$h(x)$	m	C
$\dfrac{\sin x}{2\pi x(1-x^2/\pi^2)}$	2	0.7056	$\dfrac{1-\cos x}{\pi x^2}$	3	0.7664
	3	0.7047		4	0.7658
	4	0.7029		5	0.7281
	5	0.7008			
	6	0.7005			

Одним из возможных способов повышения точности приближения $p \approx p_m$ является включение в выражение для p_m нескольких *различных* порождающих ядер из Λ_1. Например, можно рассматривать адаптивные ядра вида

$$p(x) = \frac{1}{2}\sum_{j=1}^{k}\sum_{i=1}^{m_j} w_{i,j}(h_j(x+a_{i,j}) + h_j(x-a_{i,j})), \quad \hat{p}(t) = \sum_{j=1}^{k} h_j(t)\sum_{i=1}^{m_j} w_{i,j}\cos(a_{i,j}t), \tag{8}$$

где $k \geqslant 1$ — число различных *типов* порождающих ядер, $m_j \geqslant 1$ — количество компонент порождающего ядра j-го типа, $h_j(t)$ — произвольные функции из Λ_1, $a_{i,j}, w_{i,j}$, $i = \overline{1, m_j}$, $j = \overline{1, k}$ — произвольные вещественные числа, обеспечивающие существование положительного корня χ_p уравнения (5). При этом общее число компонент ядра равняется $m_1 + \ldots + m_k \equiv m$, а число параметров — $2m$ (однако, следует отметить, что число *свободных* параметров при этом равняется $2m-1$, поскольку адаптивное ядро из неравенства сглаживания Золотарева инвариантно относительно умножения на положительное число, и один из весов можно зафиксировать). В качестве $h_j(x)$ можно использовать, например, следующие функции (символом $J_{3/2}(\cdot)$ здесь обозначена функция Бесселя):

$h_j(x)$	$\hat{h}_j(t)$
$\dfrac{1-\cos x}{\pi x^2}$	$(1-\lvert t\rvert)^+$
$\dfrac{\sin x}{2\pi x(1-x^2/\pi^2)}$	$\cos^2\dfrac{\pi t}{2}\mathbf{1}(\lvert t\rvert \leqslant 1)$
$\dfrac{15J_{3/2}^2\left(\frac{x}{2}\right)}{\lvert x\rvert^3}$	$(1-5t^2+5\lvert t\rvert^3-\lvert t\rvert^5)^+$

Для предельного значения $D(0+) = \lim_{\varepsilon\to 0+} D(\varepsilon)$ В. М. Золотарев [1] получил следующее выражение:

$$D(0+) = \sqrt{\frac{2}{\pi}}\cdot\frac{k\lambda W(\lambda)}{4V(\lambda) - ||p||\lambda}, \tag{9}$$

где $k = 4\sup_{x>0}\{(\cos x - 1 + x^2/2)/x^3\} \approx 0.396648$, λ — положительный корень уравнения

$$V(\lambda) + \frac{|\hat{p}(0)|}{12k} = W(\lambda),$$

$$W(\lambda) = \frac{(4V(\lambda) - ||p||\lambda)(V(\lambda) + \lambda^2 p^+(\lambda))}{4\lambda^2 p^+(\lambda)}, \quad V(\lambda) = \lambda\int_0^\lambda p^+(u)du.$$

Изучение поведения $D(0+)$ на ядрах вида (7), (8) представляет отдельный интерес, поскольку, с одной стороны, это позволит получить наилучший результат, возможный в рамках рассматриваемого метода, а с другой, — если подтвердится гипотеза Золотарева, приведет нас к уточнению оценки абсолютной постоянной в неравенстве Берри–Эссеена. При этом вопрос о том, чей же метод, Правитца или Золотарева, приведет к более точной верхней оценке асимптотического значения $\lim_{\varepsilon\to 0} C_\varepsilon$ на данный момент остается открытым.

В заключение автор выражает искреннюю признательность В. Ю. Королёву за постоянное внимание к работе.

Работа выполнена при поддержке Российского фонда фундаментальных исследований, проекты 08-01-00563, 08-01-00567 и 08-07-00152, Совета по грантам президента Российской Федерации для поддержки молодых российских ученых – кандидатов наук №МК-654.2008.1, а также грантом УМНИК.

Список литературы

[1] В. М. Золотарёв. Абсолютная оценка остаточного члена в центральной предельной теореме. – *Теория вероятн. и ее примен.*, 1966, т. 11, вып. 1, с. 108-119.

[2] А. Н. Колмогоров. Некоторые работы последних лет в области предельных теорем теории вероятностей. – *Вестник Моск. ун-т а*, 1953, № 10, с. 29-38.

[3] В. Ю. Королев и И. Г. Шевцова. О точности нормальной аппроксимации. I. — *Теория вероятн. и ее примен.*, 2005, т. 50, вып. 3, с. 353-366.

[4] В. Ю. Королев и И. Г. Шевцова. О точности нормальной аппроксимации. II. — *Теория вероятн. и ее примен.*, 2005, т. 50, вып. 4, с. 555-563.

[5] В. К. Мацкявичюс. О нижней оценке скорости сходимости в центральной предельной теореме. — *Теория вероятн. и ее примен.*, 1983, т. 28, вып. 3, с. 565-569.

[6] С. В. Нагаев, В. И. Чеботарев. Новый подход к оценке абсолютной константы в неравенстве Берри—Эссеена. — *Тез. докл. XXXI Дальневосточной школы-семинара им. акад. Е. В. Золотова*, Владивосток, 2006, с. 19.

[7] Б. А. Рогозин. Одно замечание к работе Эссеена "Моментное неравенство с применением к центральной предельной теореме". – *Теория вероятн. и ее примен.*, 1960, т. 5, вып. 1, с. 125-128.

[8] В. Феллер. *Введение в теорию вероятностей и ее приложения. Т. 2.* "Мир", Москва, 1967.

[9] И. Г. Шевцова. Уточнение верхней оценки абсолютной постоянной в неравенстве Берри–Эссеена. — *Теория вероятн. и ее примен.*, 2006, т. 51, вып. 3, с. 622-626.

[10] И. С. Шиганов. Об уточнении верхней константы в остаточном члене центральной предельной теоремы. – *Проблемы устойчивости стохастических моделей. Труды ВНИИСИ*, 1982, с. 109-115.

[11] P. van Beek. Fourier-analytische Methoden zur Verscharfung der Berry–Esseen Schranke. – *Doctoral dissertation*, Friedrich Wilhelms Universitat, Bonn, 1971.

[12] P. van Beek. An application of Fourier methods to the problem of sharpening the Berry–Esseen inequality. – *Z. Wahrsch. verw. Geb.*, 1972, Bd. 23, p. 187-196.

[13] H. Bergström. On the central limit theorem in the case of not equally distributed random variables. – *Skand. Aktuarietidskr.*, 1949, vol. 33, p. 37-62.

[14] A. C. Berry. The accuracy of the Gaussian approximation to the sum of independent variates. – *Trans. Amer. Math. Soc.*, 1941, vol. 49, p. 122-139.

[15] C.-G. Esseen. On the Liapunoff limit of error in the theory of probability. – *Ark. Mat. Astron. Fys.*, 1942, vol. A28, No. 9, p. 1-19.

[16] C.-G. Esseen. A moment inequality with an application to the central limit theorem. – *Skand. Aktuarietidskr.*, 1956, vol. 39, p. 160-170.

[17] P. L. Hsu. The approximate distributions of the mean and variance of a sample of independent variables. – *Ann. Math. Statist.*, 1945, Vol. 16, No. 1, p. 1-29.

[18] H. Prawitz. Ungleichungen für den absoluten Betrag einer charakteristischen funktion. — *Skand. AktuarTidskr.*, 1973, No. 1, pp. 11-16.

[19] H. Prawitz. On the remainder in the central limit theorem. — *Scand. Actuarial J.*, 1975, No. 3, pp. 145-156.

[20] K. Takano. A remark to a result of A. C. Berry. – *Res. Mem. Inst. Math.*, 1951, vol. 9, No. 6, p. 4.08-4.15.

[21] N. G. Ushakov. *Selected Topics in Characteristic Functions.* — VSP, Utrecht, 1999.

[22] D. L. Wallace. Asymptotic approximations to distributions. – *Ann. Math. Statist.*, 1958, Vol. 29, p. 635-654.

[23] V. M. Zolotarev. A sharpening of the inequality of Berry–Esseen. – *Z. Wahrsch. verw. Geb.*, 1967, Bd. 8, p. 332-342.

УДК 004.932.2

АДАПТИВНЫЙ МЕТОД ОЦЕНКИ ДВИЖЕНИЯ В ВИДЕО

© 2008 г. К. А. Симонян, С. В. Гришин, Д. С. Ватолин

{simonyan, sgrishin, dmitriy}@graphics.cs.msu.ru

Кафедра Автоматизации систем вычислительных комплексов,
Лаборатория Компьютерной графики и мультимедиа

1 Введение

В настоящее время оценка движения (ОД) широко применяется в сжатии и обработке видео. У обеих областей применения есть своя специфика.

Поскольку соседние кадры видеопоследовательности, как правило, очень похожи, можно существенно повысить эффективность алгоритмов сжатия видео за счет кодирования лишь разницы между соседними кадрами (межкадровой разницы), а не каждого кадра по отдельности. В настоящее время с целью повышения эффективности кодирования видеокодеки сжимают не межкадровую разницу, а скомпенсированную. Скомпенсированной разницей называют разницу между текущим и скомпенсированным кадрами. Скомпенсированный кадр аппроксимирует текущий кадр и построен из фрагментов предыдущего кадра с использованием информации о движении между этими кадрами. Таким образом, ОД имеет решающее значение для повышения эффективности алгоритмов сжатия видео. При этом основное требование, предъявляемое к алгоритму ОД, — вычислить параметры движения, минимизирующие скомпенсированную разницу. Таким образом, параметры движения могут не соответствовать истинному движению объектов в видеопоследовательности.

Другой сферой применения ОД является ее использование в алгоритмах обработки видео. ОД играет важную роль в таких задачах, как преобразование частоты кадров (ПЧК), сегментация видео (выделение и сопровождение объектов в кадре), преобразование чересстрочной развертки в прогрессивную, шумоподавление, восстановление сжатого видео, повышение качества видео, оцифрованного со старых кинопленок, а также во многих других. Например, использование информации о движении при шумоподавлении позволяет производить обработку не только в пространственной области, но и во временной. В ПЧК информация о движении используется для определения положения объектов на вычисляемых в процессе обработки кадрах. Очевидно, что при этом, в отличие от алгоритмов сжатия видео, критическое значение имеет правильность определения параметров движения.

2 Основные методы оценки движения. Методы сопоставления блоков

Можно выделить несколько основных групп методов ОД: методы оптического потока (optical flow) [4], методы фазовой корреляции (phase correlation) [5], методы сопоставления блоков (block matching algorithms) [6].

Наиболее широко на практике применяются методы сопоставления блоков из-за простоты аппаратной реализации и высокой вычислительной эффективности. Общая схема работы этих методов такова:

1. текущий кадр разбивается на множество непересекающихся блоков;
2. для каждого блока текущего кадра производится поиск наиболее похожего блока (реперного блока) в предыдущем кадре.

Разность между позициями текущего и реперного блоков называют вектором движения текущего блока (Рис. 1).

предыдущий кадр текущий кадр

Рис. 1: Текущий и реперный блоки и соответствующий вектор движения

В современных алгоритмах сопоставления блоков для ускорения поиска вектора движения блока используется сходство между векторами движения в пространстве и во времени. Ускорение достигается за счет перебора лишь небольшого числа векторов движения вместо проверки всех возможных. Поиск вектора движения для каждого блока в таких алгоритмах производится следующим образом:

1. формируется множество векторов-кандидатов;
2. в нем осуществляется поиск лучшего вектора-кандидата; (*)
3. производится его уточнение.

Классические BMA-методы имеют определенные недостатки, ограничивающие их применение в алгоритмах обработки видео:

- неспособность вычислить истинное движение блоков в однородных областях;
- использование блоков постоянного размера приводит либо к недостаточной точности поля векторов на границах объектов (в случае применения блоков большого размера), либо к высокой чувствительности к шуму и уменьшению скорости обработки (в случае использования блоков малого размера).

3 Разработанный алгоритм

В настоящей работе описывается метод оценки истинного движения, использующий адаптивно меняющийся размер блока и адаптивную стратегию поиска вектора движения, зависящую от контрастности блока. Общая схема алгоритма совпадает со схемой (*), каждый из этапов которой подробно описан в трех следующих подразделах. В отдельном подразделе описан метод определения размера блока.

3.1 Формирование множества векторов-кандидатов

Множество векторов-кандидатов состоит из следующих векторов (Рис. 2):

- векторы движения соседних блоков текущего кадра;
- векторы движения блоков предыдущего кадра;
- нулевой вектор, если блок достаточно контрастен.

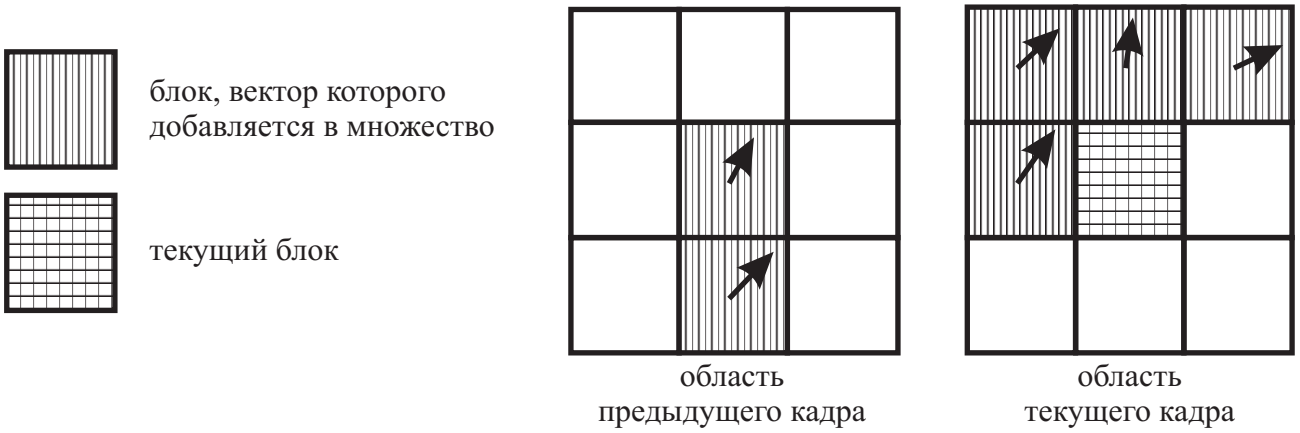


Рис. 2: Множество векторов-кандидатов

3.2 Поиск лучшего вектора-кандидата

Поиск лучшего вектора движения во множестве векторов-кандидатов производится следующим образом:

1. для каждого вектора из множества кандидатов удаляются векторы, достаточно близкие к нему по норме L_1. При этом подсчитывается число вхождений этих векторов в множество кандидатов (частота появления кандидата);

2. для каждого вектора из множества кандидатов вычисляется значение функции ошибки. В качестве функции ошибки обычно используется SAD (Sum of Absolute Differences, сумма абсолютных разностей):

$$SAD\left(t, B, \overrightarrow{d^B}\right) = \sum_{(x,y)\in B} \left|I(t,x,y) - I\left(t-1, x+d_x^B, y+d_y^B\right)\right|,$$

   где $\overrightarrow{d^B}$ — вектор движения блока B,

   $I(t,x,y)$ — яркость точки (x,y) кадра с номером t;

3. ошибка каждого вектора-кандидата умножается на его вес $w \in (0;1]$. Вес зависит от частоты появления соответствующего кандидата и контрастности блока:

   - если блок достаточно контрастный, то вес всех векторов-кандидатов полагается равным единице;
   - в противном случае (блок однородный) вес каждого вектора-кандидата обратно пропорционален его частоте появления;

4. в множестве кандидатов производится поиск лучшего вектора-кандидата — вектора с наименьшей взвешенной ошибкой.

Использование весов позволяет повысить качество работы алгоритма по сравнению с обычным способом выбора вектора из множества кандидатов, в котором учитываются только ошибки векторов. В данном случае повышение качества состоит в том, что векторное поле на выходе алгоритма больше соответствует истинному движению между кадрами, что подтверждается результатами сравнения (см. разд. 4).

3.3 Уточнение лучшего вектора-кандидата

Уточнение лучшего вектора-кандидата производится с помощью следующего итеративного процесса. Вычисляются значения функции ошибки для векторов, достаточно близких по норме L_1 к лучшему вектору, полученному на предыдущем шаге. Из них выбирается вектор, дающий наименьшее значение ошибки, после чего, в случае необходимости, операция повторяется для него. Стратегия уточнения вектора выбирается адаптивно и зависит от контрастности блока. При этом для менее контрастных блоков максимальное количество итераций уточнения меньше, чем для более контрастных блоков.

Благодаря адаптивной схеме поиска вектора движения достигается повышение качества поля векторов движения в однородных областях (Рис. 7).

3.4 Разбиение блока

Разбиение блока на четыре подблока производится адаптивно:

- в областях со сложным движением;
- в областях, где текущий размер блока не может обеспечить приемлемый уровень ошибок.

Для поиска областей со сложным движением вычисляется дисперсия координат векторов, входящих в множество векторов-кандидатов для блока. Области со сложным движением определяются путем сравнения дисперсии с порогом: если она превышает порог, то в окрестности блока имеет место сложное движение.

Приемлемый уровень ошибок для вектора движения блока определяется следующим образом. Согласно [2], имеет место зависимость между величиной

$$VAR\,(t,B) = \frac{1}{4} \sum_{i,j\in\{-2;2\}} \sum_{(x,y)\in B} |I(t,x,y) - I\,(t,x+j,y+i)|$$

и средним значением ошибки вектора. Таким образом, вычислив величину VAR для блока, становится возможным адаптивно определить порог для ошибки вектора, в случае превышения которого блок необходимо разбить на блоки меньшего размера.

Адаптивный выбор размера блока позволяет значительно повысить точность векторного поля (например, на границах объектов), не повышая при этом чувствительность к шуму [2]. Иными словами, блоки малого размера используются только в тех областях кадра, где это необходимо.

4 Сравнение

Было проведено сравнение разработанного метода с алгоритмами FAME [1] и E3DRS [3] как с одними из наиболее популярных алгоритмов ОД того же класса. Для сравнения использовался стандартный набор видеопоследовательностей. Были рассмотрены два сценария применения ОД:

1. использование блоков размера 16×16.
   Данный сценарий подразумевает разбиение области кадра только на блоки размера 16×16;

2. использование блоков размера 4×4.
   В данном случае сравнение с алгоритмом FAME не проводилось, поскольку он предназначен только для работы с блоками размера 16×16. Алгоритм E3DRS работает с блоками произвольного *фиксированного* размера, поэтому для него использовалось разбиение области кадра на блоки размера 4×4. Поскольку разработанный алгоритм способен *адаптивно* определять размер блока (см. разд. 3.4), для него использовалось начальное разбиение области кадра на блоки размера 16×16, а во время работы алгоритма происходило адаптивное разбиение блоков на подблоки по схеме $16 \times 16 \to 8 \times 8 \to 4 \times 4$.

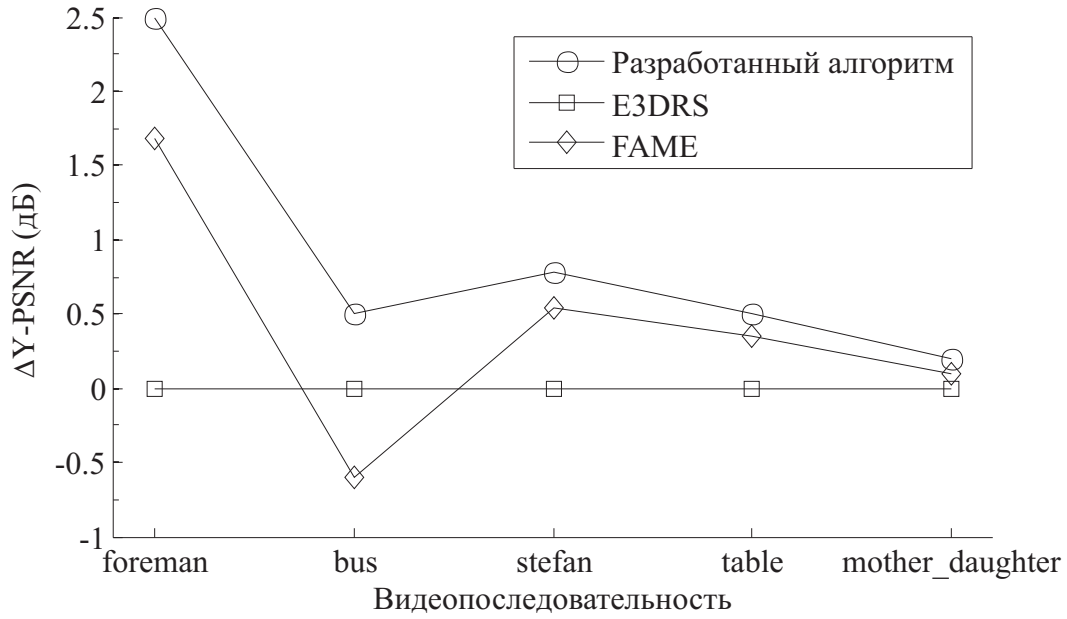


Рис. 3: Сравнение ΔY-PSNR для различных алгоритмов (размер блоков 16×16)

В сравнении применялась метрика Y-PSNR, вычисляемая для каждой пары скомпенсированный/исходный кадр по их яркостным компонентам. Большие значения метрики соответствуют меньшей скомпенсированной разнице. Иными словами, чем выше показатель Y-PSNR, тем выше точность найденных векторов движения. Для удобства сравнения была также вычислена метрика ΔY-PSNR, которая представляет собой разность значений метрики Y-PSNR сравниваемого алгоритма и алгоритма E3DRS. Также было проведено сравнение скорости обработки кадров указанными методами.

При использовании первого сценария разработанный алгоритм демонстрирует лучшие значения Y-PSNR — соответствующая ветвь на графике (Рис. 3) лежит выше остальных. Несмотря на некоторое отставание разработанного алгоритма по скорости обработки (Рис. 4), он, тем не менее, осуществляет обработку кадров видеопотока с высокой скоростью.

Как видно из Рис. 5, при использовании второго сценария разработанный алгоритм опережает алгоритм E3DRS по метрике Y-PSNR на всех видеопоследовательностях, кроме mother_daughter. Однако, субъективное сравнение (Рис. 7) демонстрирует преимущество описанного метода на этой видеопоследовательности. На приведенном кадре легко видеть неверно найденные алгоритмом E3DRS векторы движения в однородных областях (они минимизируют скомпенсированную разницу, но истинному движению не соответствуют). Поле векторов движения, полученное с помощью разработанного алгоритма, в этих областях найдено верно. Как видно, размер блоков уменьшается на границах объектов, что подтверждает эффективность работы этапа разбиения блоков. По скорости обработки (Рис. 6) разработанный метод опережает E3DRS на видео, где присутствуют достаточно большие однородные области, поскольку в этих областях используются блоки максимального размера. На видео, где таких областей нет, описанный метод уступает по скорости E3DRS. Однако скорость остается приемлемой для использования алгоритма в целях обработки видео в реальном времени.

5 Результаты

Разработан адаптивный алгоритм оценки истинного движения, не имеющий некоторых недостатков, присущих алгоритмам этого типа, что подтверждается результатами объективного и субъективного сравнений. Тем не менее, остается большое пространство для улучшения предложенного алгоритма. Дальнейшее повышение точности планируется достичь за счет использования многокадровой ОД и фильтрации поля векторов движения во временной

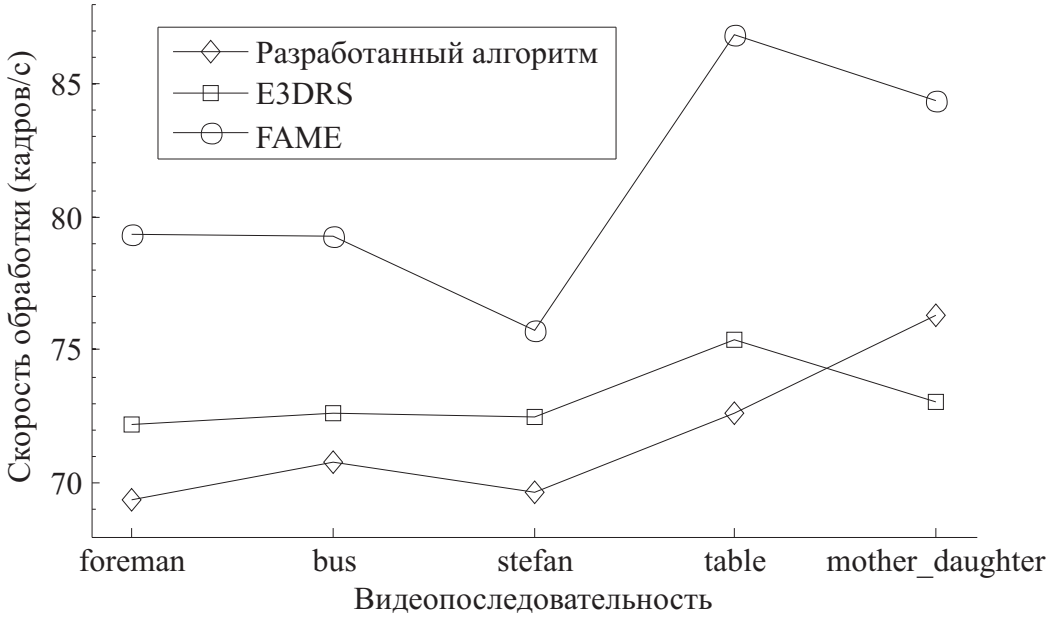


Рис. 4: Сравнение скорости обработки для различных алгоритмов (размер блоков 16×16)

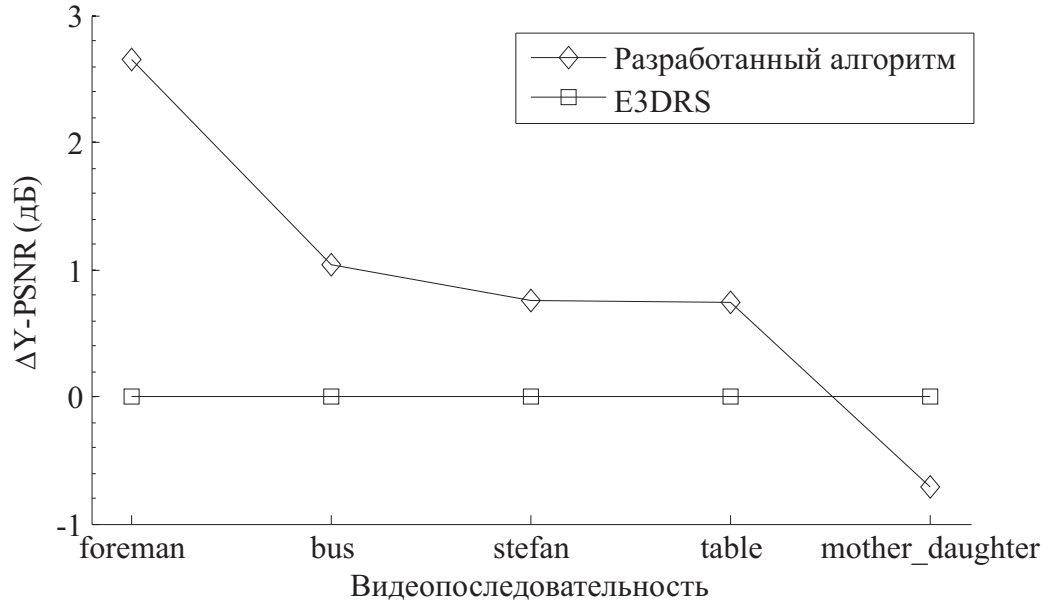


Рис. 5: Сравнение ΔY-PSNR для различных алгоритмов (размер блоков 4×4)

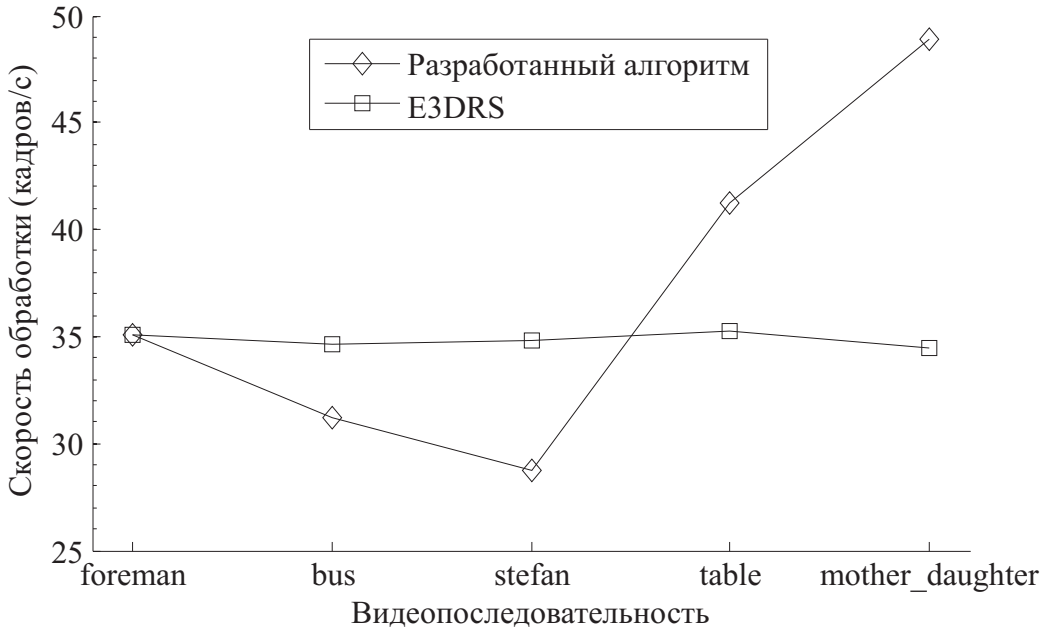
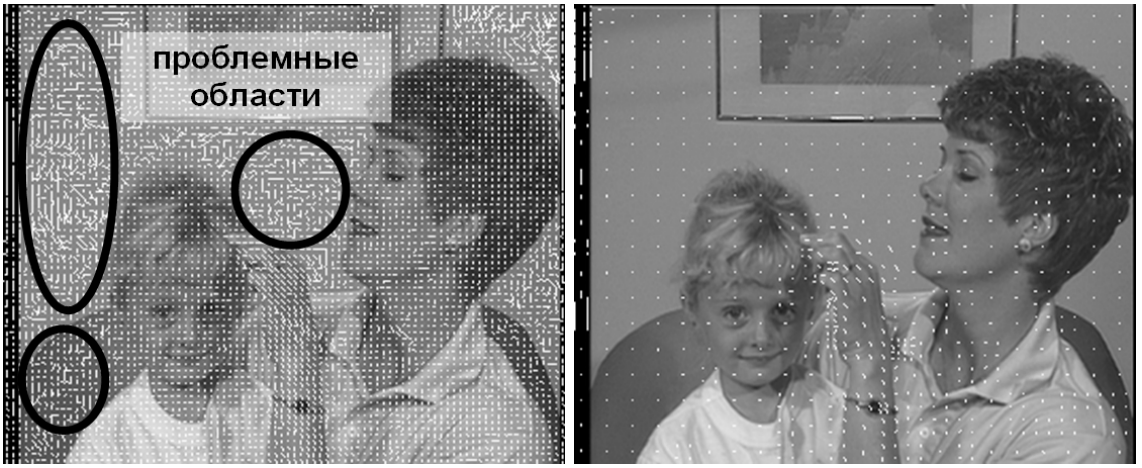


Рис. 6: Сравнение скорости обработки для различных алгоритмов (размер блоков 4×4)



поле векторов движения, полученное алгоритмом E3DRS

поле векторов движения, полученное разработанным алгоритмом

Рис. 7: Визуальное сравнение разработанного алгоритма с E3DRS (кадр видеопоследовательности mother_daughter)

области. В многокадровой ОД повышение качества достигается за счет возможности поиска реперных блоков не только в предыдущем кадре, но и в нескольких других. В ряде случаев это является единственным способом получить правильные векторы движения. В частности, многокадровая ОД позволяет корректно вычислить векторы движения в областях, где движущиеся объекты перекрывают друг друга. Анализ изменения поля векторов движения во времени позволяет выявить ошибочные векторы движения, которые затем могут быть скорректированы (фильтрация поля векторов движения).
Разработанный алгоритм реализован в виде фильтра для программы обработки видео VirtualDub. Фильтр свободно доступен по адресу http://www.compression.ru/video/motion_estimation/index.html.

Список литературы

[1] I. Ahmad, W. Zheng, J. Luo, and M. Liou. *A fast adaptive motion estimation algorithm.* IEEE Transactions on CSVT, March 2006, pp. 420–438.

[2] R. Braspenning and G. de Haan. *Efficient Motion Estimation with Content-Adaptive Resolution.* In Proceedings of ISCE'02, Sep. 2002, pp. E29–E34.

[3] S. Olivieri, L. Albani, and G. de Haan. *A low-complexity motion estimation algorithm for H.263 video coding.* In Proceedings of Philips Conference on DSP, Veldhoven, Nov. 1999, paper 17.3.

[4] E. Memin and P. Perez. *Dense estimation and object-based segmentation of the optical flow with robust techniques.* IEEE Transactions on Image Processing, 7(5):703–719, 1998.

[5] M. Li, M. Biswas, S. Kumar, and Nguyen Truong. *DCT-based phase correlation motion estimation.* In Proceedings of ICIP 2004, Vol. 1, pp. 445–448.

[6] С. Путилин. *Быстрый алгоритм нахождения движения в видеопоследовательностях.* Труды конференции Graphicon-2006, с. 407–410, Новосибирск, Академгородок, 1–4 Июля 2006.

УДК 519.683.8

ИМПОРТ ВЫЧИСЛИТЕЛЬНОЙ МОДЕЛИ ЯЗЫКА SCHEME В ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ОКРУЖЕНИЕ

© 2008 г. А. В. Столяров

avst@cs.msu.ru

Кафедра Алгоритмических языков

1 Введение

Метод непосредственной интеграции впервые предложен в статье [1] в качестве подхода к интеграции разнородных языковых изобразительных средств в одном проекте. Методу также посвящены работы [2], [3], [4], [6] и [7].

В основе метода непосредственной интеграции лежит моделирование средствами базового языка одновременно вычислительной модели (семантики) и изобразительных конструкций (синтаксиса) альтернативного языка. При этом синтаксис альтернативного языка моделируется близкими по визуальному восприятию арифметическими выражениями, записываемыми в соответствии с правилами языка базового. Это достигается введением объектов, соответствующих понятиям альтернативного языка и переопределением символов инфиксных операций для таких объектов; ясно, что для применения метода необходимо наличие в базовом языке соответствующих возможностей.

Метод практически реализован в библиотеке InteLib, которая предоставляет набор классов языка C++, реализующих S-выражения [8] как гетерогенные структуры данных. Вводимый для этих классов набор инфиксных операций позволяет записывать конструкции, являющиеся с точки зрения компилятора C++ арифметическими выражениями, которые синтаксически близки традиционным для языка Lisp S-выражениям. Второй слой библиотеки позволяет производить *вычисления* этих выражений, достигая, таким образом, их семантической эквивалентности соответствующим выражениям языка Lisp.

2 Проблемы реализации модели языка Scheme

Исходно вычисления конструкций языка Lisp в библиотеке InteLib были реализованы простейшим для понимания способом, а именно, с использованием рекурсивного вызова программы вычисления выражения. Так, например, при вычислении

```
(+ 1 (* 2 x) (* 3 x x))
```

вычислитель сначала вызывался для всего выражения целиком. Поскольку для передачи управления функции «+» необходимо сначала вычислить её арнументы, вычислитель вызывал рекурсивно сам себя для вычисления (поочерёдно) выражений `1`, `(* 2 x)` и `(* 3 x x)`. Будучи вызванным для константы `1`, он немедленно возвращал управление, поскольку действие по вычислению константы является элементарным. Будучи далее вызванным для `(* 2 x)`, вычислитель вновь рекурсивно вызывал сам себя уже для выражений `2` и `x`; в первом случае управление возвращалось немедленно, во втором — анализировался текущий лексический контекст, и возвращалось значение, лексически связанное с `x` (то есть локальное), если таковое присутствовало, в противном случае возвращалось динамическое (глобальное) значение `x`. Получив все значения, необходимые для вызова функции «*», вычислитель вызывал тело функции, предварительно сформировав вектор из её фактических параметров, получал значение, возвращал управление, и т. д.

В первой (пробной) версии библиотеки InteLib, написанной в 1999 году, результат работы вычислителя возвращался как значение функции. Для оптимизации остаточной рекурсии, а также для реализации формы `SETF` механизм возврата значений был усовершенствован уже во

второй версии: в функцию, вычисляющую S-выражение, стал передаваться дополнительный параметр, представляющий собой объект специального класса, который отвечал за переключение лексических контекстов, восстановление значений динамически связываемых переменных и хранение возвращаемого значения, причём возврат результата вычисления стало возможно производить в соответствии с одной из трёх моделей: вернуть обычное значение, вернуть *ссылку* на имеющуюся структуру данных (например, на левую или правую часть существующей точечной пары), либо вернуть выражение, которое в свою очередь подлежит вычислению. Вторая из перечисленных моделей сделала возможной реализацию `SETF`, третья — оптимизацию остаточной рекурсии. Так или иначе, для вычисления аргументов функции, а также для вызова других функций в функциях высоких порядков (таких как `MAPCAR` или `SORT`) вычислитель продолжал делать рекурсивные вызовы самого себя.

Для вычислительной модели языка Scheme такая реализация оказывается неприемлема из-за наличия в этом языке механизма *продолжений* (continuations), реализуемого функцией `CALL-WITH-CURRENT-CONTINUATION` (или просто `CALL/CC`). С теоретической точки зрения *продолжение* представляет собой *последовательность действий, которые необходимо выполнить до завершения вычислений.* Каждое элементарное действие, такое как вычисление константы или применение функции к вычисленным аргументам, рассматривается как *переход* от одного *продолжения* к другому; иногда также говорят, что новое (очередное) продолжение возникает в тот же момент, когда начинается выполнение очередного элементарного вычисления, и *ожидает значения.* Так, например, при вычислении

```
(+ (* a 15) (* b 25))
```

первое *продолжение* имеет вид «вычислить выражение целиком»; от него осуществляется переход к *продолжению* «получить значение, затем вычислить `(* b 25)`, затем к двум последним значениям применить `+`». Можно сказать, что это второе *продолжение* возникает в тот же момент, когда начинается вычисление `(* a 15)`, и ожидает его результата.

Функция `CALL/CC` позволяет «законсервировать» текущее *продолжение*, с тем чтобы позже можно было выполнить те же действия, но начальный результат дать другой.

Если рассматривать описанную выше простую рекурсивную реализацию вычислителя, окажется, что информация, составляющая *продолжение*, рассеяна по локальным переменным многих одновременно активных экземпляров функции-вычислителя, и, более того, часть информации вообще не является явно доступной: действительно, информация о предстоящей последовательности действий оказывается представлена совокупностью адресов возврата, сохранённых в стеке, которые в явном виде программисту недоступны, во всяком случае, если говорить о штатных переносимых средствах. Известна реализация механизма *продолжений* для языка C, основанная на недокументированных особенностях функций `setjmp()` и `longjmp()` (см. [11]), однако гарантировать переносимость такого решения невозможно.

3 Идея применённого решения

Как можно заметить, при работе с *продолжением* речь идёт о двух основных сущностях: о действиях, которые необходимо выполнить, и о сохранённых значениях, которые являются результатами уже выполненных действий.

Легко видеть также, что при выполнении вычислений бóльших уровней вложенности одни действия по мере их выполнения могут заменяться другими либо преобразовываться в готовые значения, причём чем позднее было получено значение, тем раньше оно понадобится при вызове очередной функции; аналогично, чем позднее в список невыполненных действий попала та или иная операция, тем раньше настанет момент, когда её придётся выполнить.

Получается, что *продолжение* можно представить в виде двух стеков: в одном будем сохранять операции, которые необходимо произвести, во втором — значения, которые позднее станут аргументами операций. Для примера рассмотрим процесс вычисления выражения `(+ (* a 15) (* b 25) 7)` при значениях переменных `a` и `b`, соответственно, `10` и `100`. Последовательность шагов (*продолжений*) такого вычисления показана в табл. 1.

При этом мы обозначаем словом `EVAL` операцию, состоящую в вычислении заданного выражения, а словом `CALL` — применение заданной функции к заданному количеству аргументов.

№ шага	Стек значений	Стек операций
1	∅	`EVAL (+ (* a 15) (* b 25) 7)`

2	∅	`EVAL (* a 15)`
		`EVAL (* b 25)`
		`EVAL 7`
		`CALL/3 +`

3	∅	`EVAL a`
		`EVAL 15`
		`CALL/2 *`
		`EVAL (* b 25)`
		`EVAL 7`
		`CALL/3 +`

4	10	`EVAL 15`
		`CALL/2 *`
		`EVAL (* b 25)`
		`EVAL 7`
		`CALL/3 +`

5	15
	10
		`CALL/2 *`
		`EVAL (* b 25)`
		`EVAL 7`
		`CALL/3 +`

6	150	`EVAL (* b 25)`
		`EVAL 7`
		`CALL/3 +`

7	150	`EVAL b`
		`EVAL 25`
		`CALL/2 *`
		`EVAL 7`
		`CALL/3 +`

8	100
	150
		`EVAL 25`
		`CALL/2 *`
		`EVAL 7`
		`CALL/3 +`

9	25
	100
	150
		`CALL/2 *`
		`EVAL 7`
		`CALL/3 +`

10	2500 150	`EVAL 7` `CALL/3 +`
11	7 2500 150	`CALL/3 +`
12	2657	∅

Таблица 1: Пример последовательности *продолжений*

Первая операция выполняется за один шаг для констант и переменных, т.е. операция убирается из стека действий, а в стек результатов добавляется значение. Для вызовов функций операция `EVAL` выполняется иначе: сама она из стека действий убирается, а вместо неё заносится операция `CALL` для соответствующего количества параметров и операции `EVAL` для вычисления каждого аргумента функции; стек результатов при этом не меняется. Наконец, операция `CALL` извлекает из стека результатов соответствующее количество параметров, применяет к ним заданную функцию, а затем результат применения помещает снова в стек результатов.

Шаги, перечисленные в таблице, соответствуют в теоретической терминологии последовательности *продолжений*. Вычисления считаются оконченными, когда стек действий пуст, то есть *продолжение* с пустым стеком действий считается конечным. К этому моменту в сте-

ке результатов должно остаться ровно одно значение, которое и объявляется окончательным результатом вычисления.

Ясно, что в сохранении всех промежуточных *продолжений* нет никакого практического смысла, если только не выполняются функции, «консервирующие» текущее *продолжение*. Таким образом, достаточно иметь *один* экземпляр стеков и изменять их, а при необходимости для целей «консервирования» создавать копии.

При реализации в объектно-ориентированном окружении естественной выглядит инкапсуляция обоих стеков в класс, который и называется `Continuation` (т.е. «продолжение»), причём такой объект должен иметь методы по добавлению новых заданий, по извлечению результатов, а также метод, выполняющий *один шаг вычислений*, или, иначе говоря, *переход к следующему продолжению*.

Объект такого класса может при желании рассматриваться и как своего рода виртуальная машина, исполняющая программы на языке Scheme.[1]

Поскольку объект, представляющий *продолжение*, неизбежно должен существовать и быть доступным во всём коде, имеющем отношение к вычислению S-выражений (исполнению программы на Scheme), на этот объект можно также возложить некоторые вспомогательные функции, такие как управление связыванием локальных переменных (иначе говоря, управление «лексическими контекстами») и поддержку различных моделей возврата значения из функции.

4 Рабочая реализация

4.1 Предварительные замечания

4.1.1 Структура библиотеки InteLib

Библиотека InteLib спроектирована в несколько слоёв, причём нижний слой представляет собой S-выражения, воспринимаемые как гетерогенные структуры данных в отрыве от каких-либо вычислительных моделей [6].

Верхний слой библиотеки содержит подсистемы, относящиеся к вычислительным моделям различных языков. В настоящее время дистрибутив библиотеки содержит модели языков Lisp, Scheme и Refal, причём последняя пока находится в стадии экспериментального кода. Ведутся также работы над моделями языков Planner, Prolog, Datalog и Erlang. Необходимо отметить общее для всех этих языков свойство: все они используют гетерогенные структуры данных, основанные на односвязных списках, т.е. S-выражения, хотя при описании этих языков термин «S-выражение» почему-то обычно не употребляется.

Кроме верхнего и нижнего слоёв, библиотека имеет также промежуточный слой, в котором реализованы некоторые инструментальные средства для работы с S-выражениями, в частности — средства работы с потоками ввода-вывода, синтаксического анализа текстового представления S-выражений и т.п. Также в этот слой вынесены общие для различных вычислительных моделей части, имеющие отношение к *вычислению* S-выражений.

Библиотека ориентирована на работу в «чисто компилируемом» окружении и использование арифметических выражений C++, заменяющих соответствующие конструкции моделируемых языков. Тем не менее, в мультипарадигмальном окружении практически всегда возникает настойчивая потребность в написании некоторых модулей целиком в синтаксисе оригинального (импортируемого) языка. В дистрибутив библиотеки InteLib входят трансляторы для каждого из моделируемых языков программирования, позволяющие на основании текста на таком языке построить модуль на C++, состоящий из заголовочного файла и файла реализации и использующий конструкции библиотеки InteLib. Кроме того, для целей отладки в библиотеку входят и расширяемые интерактивные интерпретаторы языков Lisp и Scheme; при необходимости эти интерпретаторы можно включать и в пользовательские приложения в качестве встроенных.

[1] Естественно, здесь имеется в виду внутреннее представление программ, поскольку задачи лексического и синтаксического анализа кода относятся к совершенно иной проблемной области.

4.1.2 Соглашения об именованиях классов в библиотеке InteLib

Отметим, что в библиотеке InteLib S-выражения различных типов представляются объектами классов, имеющих общего абстрактного предка `SExpression`. Имена классов, представляющих конкретные типы S-выражений и относящихся к нижнему слою библиотеки, начинаются с префикса `SExpression`; такое же соглашение используется для классов промежуточного (инструментального) слоя.

Классы S-выражений, относящихся к конкретным вычислительным моделям (Lisp, Scheme, Refal) имеют в названии префикс, указывающий на соответствующую вычислительную модель (соответственно префиксы `LExpression`, `SchExpression` и `RfExpression`).

Работа с S-выражениями строится через систему «умных указателей» и «ведущих указателей» (*smart pointers* и *master pointers* в терминологии книги [12]). Умные указатели — это объекты, ведущие себя подобно указателям, но выполняющие какие-либо дополнительные функции; в случае библиотеки InteLib наиболее важная из таких функций — поддержка счётчиков ссылок, позволяющая автоматически уничтожать объекты S-выражений, на которые больше никто не ссылается. Кроме того, на умные указатели возлагается ряд функций, специфичных для S-выражений, таких как композиция и декомпозиция списков, конверсия во встроенные типы и т.д. Все умные указатели в библиотеке InteLib наследуются от класса `SReference`, который представляет собой основной механизм доступа к S-выражениям. Классы умных указателей, вводимые для обслуживания специфических типов S-выражений, имеют имена, полученные из названий соответствующих классов S-выражений путём удаления слова `Expression` и добавления суффикса `Ref`: например, хеш-таблицы в InteLib представляются классом `SExpressionHashTable`, а соответствующий им класс умного указателя имеет имя `SHashTableRef`; аналогично, *символ* языка Scheme представляется классом `SchExpressionSymbol`, а соответствующий класс умного указателя называется `SchSymbolRef`.

Ведущие указатели отличаются от умных указателей тем, что сами автоматически создают объект, на который указывают. В библиотеке InteLib классы ведущих указателей всегда наследуются от соответствующих умных указателей и имеют имена, полученные отбрасыванием суффикса `Ref`; так, ведущие указатели для хеш-таблиц называются `SHashTable`, а для символов языка Scheme — `SchSymbol`.

Для обработки исключительных ситуаций используется класс, называемый `IntelibX` и его потомки, названия которых формируются добавлением к префиксу `IntelibX` слов, обозначающих конкретную ошибочную ситуацию, причём эти слова набираются в нижнем регистре через символ подчёркивания; например, арифметические функции при подаче им нечисловых аргументов выбрасывают исключение `IntelibX_not_a_number`.

Классы, не относящиеся к трём вышеописанным иерархиям, т.е. не являющиеся ни S-выражением, ни умным либо ведущим указателем S-выражения, ни классом ошибок, именуются с обязательным использованием префикса `Intelib` (для нижнего слоя библиотеки), либо префиксов `Lisp`, `Scheme`, `Refal` для соответствующих подсистем. Так, класс синтаксического анализатора текстового представления S-выражений называется `IntelibReader`, а его потомок, адаптированный под конкретику кода на Scheme, называется `SchemeReader`.

В соответствии с этими соглашениями класс, соответствующий понятию *продолжения*, изначально назывался `SchemeContinuation`. Такого названия мы и будем придерживаться в изложении.

4.2 Основные принципы реализации класса `SchemeContinuation`

Основой реализации класса `SchemeContinuation` являются стек действий (*ToDo stack*) и стек результатов (*result stack*). Стек результатов реализован в виде массива элементов типа `SReference`, что позволяет использовать фрагменты этого стека в качестве векторов параметров при вызове функций, избегая, таким образом, выделения динамической памяти при каждом вызове Scheme-функции. Сам массив создаётся в динамической памяти при создании объекта `SchemeContinuation` и при необходимости расширяется (каждый раз вдвое). Практика показала, что такая необходимость (при начальном размере массива 256 элементов) возникает редко.

Стек действий также представляет собой динамически расширяемый массив, но его элементы имеют более сложную структуру. Каждый элемент содержит код операции из некоторого предопределённого множества, опциональный параметр, а также поле для хранения отладочной информации о стеке вызовов Lisp-форм; последнее может быть исключено при компиляции библиотеки заданием соответствующей директивы условной компиляции.

Кроме этих двух стеков, в объекте хранится также указатель на текущий (активный) «лексический контекст». Класс также вводит два статических поля для поддержки механизма прерывания вычислений.

4.3 Основные коды операций

Работа с объектом `SchemeContinuation` обычно начинается с помещения в стек действий операции «просто вычислить» с параметром, в качестве которого выступает подлежащее вычислению выражение. Соответствующий код операции — `just_evaluate`.

Вторая наиболее часто употребляемая[2] операция — вызов функции для заданного количества аргументов. Поскольку в языке Scheme сам объект вызываемой функции может быть результатом вычисления, причём такое вычисление, вообще говоря, происходит до вычисления параметров функции, соответствующая операция работает следующим образом: если `N` — заданное количество параметров, то от вершины стека отсчитывается `N` позиций, из полученной позиции извлекается объект функции, после чего для этого объекта производится вызов виртуального метода, ответственного за собственно применение функционального объекта к аргументам, причём в качестве вектора аргументов передаётся адрес позиции в стеке, следующей за позицией, из которой извлечён объект-функция. Указатель стека предварительно откатывается на `N`+1 позицию, так что результат выполнения функции будет записан в элемент массива (стека), ранее содержавший объект функции и не задействованный для хранения параметров.

Из соображений эффективности операция «вызов функции для `N` аргументов» не использует поле параметра для хранения числа `N`. Вместо этого за код именно этой операции принимается *любое неотрицательное число*, записанное в поле кода операции, т.е. число `0` соответствует вызову функции без аргументов, число `1` — вызову унарной функции и т.д. Все остальные операции имеют отрицательные коды, в частности, операция `just_evaluate` имеет код `-1`.

Кроме того, для удобства вводятся следующие команды:

- `evaluate_prepared` вычисляет свой параметр; отличается от `just_evaluate` предположением, что все аргументы уже вычислены. Применяется обычно при вычислении функций высших порядков.

- `evaluate_progn` вычисляет последовательность форм, в качестве которой рассматривается параметр; естественно, параметр должен представлять собой список. Эта операция введена для удобства и применяется, например, при выполнении обычных функций, написанных на Scheme, а также и в других случаях, когда язык предполагает выполнение последовательности форм (в предложениях формы `COND`, в теле `LET` и т.п.).

- `quote_parameter` помещает свой параметр в стек результатов. Применяется при вычислении специальных форм, которые в конце своей работыш (в ходе которой могут активно использоваться оба стека) должны возвращать заранее известный результат.

- `drop_result` убирает из стека результатов находящееся на его вершине значение. Применяется в случаях, если результат вычисления очередной формы должен быть проигнорирован; в частности, это делается для всех элементов списка, поданного команде `evaluate_progn`, кроме последнего. Параметр игнорируется.

- `return_unspecified` — помещает в стек результатов значение, соответствующее понятию *unspecified*; это значение, в качестве которого выступает специально созданный объект

[2]Заметим, пример, приведённый в таблице 1, использует только эти две команды

атомарного S-выражения, возвращается из всех функций и форм, относительно которых в стандарте Scheme указано, что возвращаемое значение не определено. Параметр игнорируется.

4.4 Реализация условных конструкций

Реализация специальных форм, осуществляющих вычисления в зависимости от результатов предыдущих вычислений (таких как `IF`, `COND`, `AND` и `OR`) оказывается нетривиальной задачей в силу вышеупомянутой недопустимости рекурсивного вызова вычислителя. Так, при вычислении формы `COND` для каждого предложения необходимо вычислить его первый элемент, и если результатом вычисления будет `#F` (логическая ложь), перейти к следующему предложению, в противном случае продолжить вычисление форм текущего предложения, а остальные предложения пропустить. Однако из кода написанной на C++ функции, ответственной за вычисление формы `COND`, мы *не имеем права произвести вычисление первой формы предложения*, как и любой другой формы. Всё, что можно сделать — это поместить те или иные данные в стеки (как в стек действий, так и, при необходимости, в стек результатов) и вернуть управление.

В процессе реализации оказалось, что введённых выше кодов операций недостаточно; вообще, рассматриваемая реализация базировалась на введении средств *ad hoc*, по мере возникновения конкретных потребностей.

4.4.1 Форма `COND`

Для реализации формы `COND` были введены два кода операций, `cond_clause` и `end_of_clauses`. Первая из них извлекает из стека результатов значение, находящееся на вершине стека (то есть последнее вычисленное). Если значение представляет собой логическую ложь, операция на этом заканчивается; в противном случае параметр операции рассматривается как список форм, подлежащих вычислению (то есть в качестве параметра задаётся остаток `COND`–предложения). Прежде чем приступить к вычислению форм из своего параметра, операция `cond_clause` изымает из стека действий все имеющиеся там действия одно за другим до тех пор, пока не обнаружит в очередном элементе стека команду `end_of_clauses` (таким образом мы избегаем вычисления оставшихся предложений формы `COND`). После этого в стек помещаются команды для последовательного вычисления форм текущего предложения (элементов списка, заданного в параметре операции), причём после каждого, кроме последнего, помещается команда `drop_result`. Случай пустого списка в параметре рассматривается как специальный; в этой ситуации вместо планирования дальнейших вычислений выполняется действие существенно более простое, а именно, значение, только что извлечённое из стека результатов, помещается обратно. Именно оно в этом случае будет конечным результатом вычисления формы.

Команда `end_of_clause` представляет собой «не-операцию» (no-op), то есть, будучи извлечённой из стека действий, не делает ровным счётом ничего. Используется она исключительно для пометки позиции в стеке действий, соответствующей концу формы `COND` или другой формы.

Используя эти два кода операций, мы можем, имея форму `COND`, запланировать её вычисление следующим образом. Сначала поместим в стек действий команду `end_of_clauses` (она, таким образом, окажется глубже всего остального, имеющего отношение к нашей форме). Далее на случай, если ни одно предложение `COND` не будет выполнено (то есть все условия при вычислении дадут значение «ложь») поместим в стек действий команду `quote_parameter` с параметром `#F`.

Далее просмотрим предложения формы `COND` в обратном порядке (в имеющейся реализации это делается на обратном ходе рекурсии) и для каждого предложения запишем в стек действий сначала команду `cond_clause` с «хвостом» предложения в качестве параметра, затем команду `just_evaluate` с головой предложения в качестве параметра. После этого можно будет вернуть управление.

Итак, дальнейшее выполнение будет происходить следующим образом. Будет извлечена из стека команда `just_evaluate`, параметром которой является первый элемент (т.е. усло-

вие) первого предложения COND; после её выполнения (которое может, разумеется, потребовать сколь угодно сложных действий, но все эти действия затронут только позиции обоих стеков, находящиеся выше рассматриваемых) на вершине стека результатов окажется логическое значение, показывающее, следует ли выполнять текущее предложение COND или перейти к следующему. Далее из стека действий будет извлечена команда cond_clause, которая этим значением воспользуется, и так для каждого предложения COND. Если одно из предложений будет выполняться (то есть его условие даст значение истины), весь остаток операций, помещённых в стек формой COND, будет из стека изъят, а затем будут вычислены формы текущего предложения, причём в стеке результатов останется только результат последней из них.

4.4.2 Формы IF, AND и OR

Механизм, подобный вышеописанному, был применён и для реализации других условных форм.

В частности, для формы IF оказалось достаточно уже введённых команд. При её вычислении в стек действий помещается сначала команда end_of_clauses; как и для формы COND, она оказывается в стеке глубже всех. Затем в зависимости от наличия или отсутствия в форме ветки *else* (т.е. третьего аргумента формы) в стек действий помещается либо команда just_evaluate с этим третьим аргументом в качестве параметра, либо команда quote_parameter с параметром «ложь». Затем в стек действий записывается команда cond_clause с параметром, представляющим собою список из одного элемента — ветки *then* (то есть второго параметра формы IF). Наконец, последним в стек записывается команда just_evaluate с параметром, равным первому параметру формы IF., после чего можно вернуть управление. Нетрудно видеть, что дальнейшие вычисления по своей семантике будут в точности соответствовать порядку действий, предполагаемому формой IF.

Форма OR реализуется просто как частный случай формы COND со всеми предложениями, состоящими только из первого элемента. Иначе говоря, в стек действий сначала помещается команда end_of_clauses, затем quote_parameter с параметром #F, а затем в обратном порядке для каждого аргумента формы — команда cond_clause с пустым списком в качестве параметра и команда just_evaluate с параметром, соответствующим рассматриваемому аргументу формы.

Чуть сложнее обстоят дела с формой AND, поскольку здесь следует прекратить дальнейшие вычисления (изъять из стека действий всё до метки end_of_clauses) не по значению «истина», а, наоборот, по значению «ложь». Поэтому для реализации формы AND пришлось ввести ещё одну операцию, которая названа bail_on_false. По своей семантике она аналогична (и в известном смысле противоположна) команде cond_clause: при её выполнении из стека результатов извлекается значение, и если оно истинно, то не делается больше ничего, если же оно ложно, то из стека действий изымаются все команды до метки end_of_clauses, а затем в стек результатов записывается значение «ложь». Команда bail_on_false не использует (игнорирует) свой параметр. Таким образом, для вычисления формы AND в стек действий необходимо поместить end_of_clauses, затем just_evaluate с последним аргументом формы AND в качестве параметра, затем для всех аргументов формы, кроме последнего, в обратном порядке — bail_on_false и just_evaluate с рассматриваемым аргументом формы в качестве параметра.

4.5 Реализация итерационных вычислений

Функции и формы, требующие итерационного исполнения и вычислений форм на каждой итерации, оказываются ещё более нетривиальны в реализации, когда прямой вызов вычислителя запрещён. К этой категории функций относятся, например, стандартные функция MAP и форма DO.

В рассматриваемой реализации для этой цели введены, во-первых, две дополнительные команды generic_iteration и iteration_callback, и, во-вторых, вспомогательный абстрактный класс SExpressionGenericIteration, наследники которого реализуют конкретные итерационные процессы. Класс вводит четыре чисто виртуальных метода: NeedAnotherIteration, ScheduleIteration, CollectResultOfIteration и ReturnFinalValue, которые необходимо

определить в классе-потомке. Метод `NeedAnotherIteration` возвращает булевское значение, показывающее, следует ли выполнять ещё одну итерацию цикла. Предполагается, что этот метод ничего не меняет в объекте и не производит никаких активных действий, а только вычисляет на основании тех или иных признаков, завершен ли итерационный процесс. Задачей метода `ScheduleIteration` является размещение в стеке действий набора команд, необходимых для выполнения очередной итерации; метод `CollectResultOfIteration` вызывается после выполнения итерации, чтобы дать возможность объекту извлечь из стека результатов итоги выполнения очередной итерации. Наконец, метод `ReturnFinalValue` вызывается после того, как `NeedAnotherIteration` вернул логическую ложь. Этот метод должен поместить в стек результатов то значение, которое будет использовано в качестве итогового результата цикла; он также может вместо готового результата поместить в стек действий инструкции по его вычислению.

Команда `generic_iteration` выполняется следующим образом. Из параметра команды извлекается объект класса `SExpressionGenericIteration` и для этого объекта вызывается метод `NeedAnotherIteration`. Если метод вернул ложь, вызывается метод `ReturnFinalValue` и работа на этом заканчивается. Если же была возвращена истина, то в стек действий заносятся сначала команда `generic_iteration` (чтобы описываемые действия были повторены после выполнения очередной итерации), затем команда `iteration_callback` с тем же объектом `GenericInteration` в качестве параметра, и, наконец, вызывается метод `ScheduleIteration` для добавления в стек действий инструкций, необходимых для выполнения очередной итерации.

Команда `iteration_callback` выполняется проще: предполагая, что её параметром является объект `SExpressionGenericIteration`, она вызывает его метод `CollectResultOfIteration`.

Таким образом, для выполнения произвольного итерационного процесса достаточно описать соответствующий класс (потомок `SExpressionGenericIteration`), сконструировать его объект и поместить в стек действий команду `generic_iteration` с этим объектом в качестве параметра.

4.6 Работа с контекстами и присваиваниями

Часто возникает потребность изменить содержимое области памяти, содержащей S-выражение (например, присвоить значение Scheme-переменной), причём новое значение, а иногда и расположение изменяемой памяти является результатом вычислений; такое случается, например, при выполнении формы `SET!`.

Следует отметить, что любое присваивание в терминах Scheme является на самом деле изменением значения объекта класса `SReference`. Для выполнения присваиваний вводится ещё один вспомогательный класс, `SExpressionLocation`, который инкапсулирует указатель на `SReference`. Для присваивания вводятся две команды, `assign_to` и `assign_location`. Обе извлекают значение из стека результатов и заносят его по адресу, заданному объектом `SExpressionLocation`, только `assign_to` берёт этот объект из своего параметра, а `assign_location` — из стека результатов, позволяя, таким образом, производить присваивание в вычисленную позицию.

Отметим, что объект `SExpressionLocation` способен, кроме собственно указателя на позицию в памяти, хранить ещё и «умную ссылку» на структуру данных, *содержащую* эту позицию. Например, если указатель указывает на car или cdr от точечной пары, то «умную ссылку» есть смысл установить на саму эту точечную пару. Это гарантирует, что соответствующая позиция не перестанет быть валидной, пока существует объект `SExpressionLocation`, указывающий на данную позицию.

Вычисление некоторых форм (например, форм семейства `LET`) предполагает манипуляцию контекстами: например, при выполнении обычной формы `LET` вычисление значений, подлежащих связыванию с переменными, производится во внешнем контексте, а для связывания создаётся ещё один (уже внутренний) контекст, в котором и производится связывание, а также и вычисление форм, образующих тело `LET`. Контекст представляется классом `SchExpressionContext`; смена контекста производится командой `set_context`, которая берёт объект контекста из параметра.

Отметим один важный момент. Обычно после вычисления формы необходимо восстановление контекста, бывшего активным до начала её вычисления. Потому при планировании вычисления формы, вводящей локальный контекст, в стек действий заносится операция `set_context` с параметром, равным текущему контексту. Если не предпринять специальных мер, такая техника может привести к расходованию стека в ситуации остаточной рекурсии и, таким образом, свести на нет эффект от оптимизации остаточной рекурсии. В связи с этим функция, отвечающая за занесение в стек действий, при попытке занесения очередной команды `set_context` проверяет, не находится ли на вершине стека другая команда `set_context`, и если это так, не производит занесения новой команды. Это не нарушает семантики вычислений, поскольку эффект от выполнения двух команд `set_context` подряд заведомо эквивалентен эффекту от выполнения последней из них (то есть находящейся глубже в стеке) и, следовательно, новая команда просто не нужна.

4.7 Поддержка моделей возврата

Возврат значений из Scheme-функций, написанных на C++, может, как уже говорилось, осуществляться по одной из нескольких моделей. Для поддержки этих моделей класс `SchemeContinuation` имеет специальные методы:

```
void RegularReturn(const SReference &ref);
void ReferenceReturn(SReference &ref, const SReference &superstruct);
void TailReturn(const SReference &ref);
```

Первый из этих методов, `RegularReturn`, предназначен для обычного возврата значения и, как правило, применяется для значения, которое только что построено. Работает метод очень просто: переданное ему значение он помещает в стек результатов.

Второй метод, `ReferenceReturn`, предназначен для возврата из функций, выбирающих подвыражение из некоего выражения. Таковы, например, функции `CAR` и `CDR`: они возвращают не новое выражение, а *часть существующего выражения*. Поведение метода ReferenceReturn зависит от текущего содержимого стека действий. В случае, если на вершине стека находится операция `assign_location` (или эта операция отделена от вершины стека операцией `set_context`, что тоже допустимо), метод `ReferenceReturn` формирует объект `SExpressionLocation` из своих аргументов. В противном случае метод работает как для простого возврата, то есть помещает свой первый аргумент в стек результатов. Наличие этого метода позволяет реализовать функцию, подобную форме `SETF` из Common Lisp.

Наконец, последний из трёх методов, `TailReturn`, предназначен для оптимизации остаточной рекурсии. Этот метод применяется в случае, если необходимо вернуть в качестве значения результат очередного вычисления. Работает метод очень просто: помещает в стек действий команду `just_evaluate`, а свой аргумент передаёт ей в качестве параметра.

5 Взаимовлияние диалектов InteLib Scheme и InteLib Lisp

После завершения первой рабочей реализации InteLib Scheme было принято решение переписать реализацию диалекта InteLib Lisp с использованием новых механизмов. Это позволило добиться повышения временно́й эффективности работы Lisp-вычислителя в несколько раз; скорее всего, основная экономия достигается за счёт использования фрагментов стека результатов в качестве вектора параметров при вызовах функций.

Перечислим основные принципиальные различия между диалектами InteLib Lisp и InteLib Scheme:

1. В Scheme вычисляются все элементы формы, если только первый элемент не является символом, с которым связана специальная синтаксическая конструкция; в диалекте Lisp первый элемент формы не вычисляется и должен представлять собой либо символ, либо лямбда-список.

2. В Scheme символ имеет только одно значение; функция, ассоцированная с символом, является значением символа. В диалекте Lisp символ имеет, независимо друг от друга, значение и ассоциированную функцию.

3. В диалекте Lisp имеются динамически связываемые символы, вводимые формой `DEFVAR`; в Scheme таких нет.

4. В диалекте Lisp роли пустого списка и логической лжи совмещены и исполняются символом `NIL`. В Scheme пустой список и логическая ложь обозначаются специальными объектами, различающимися между собой и не являющимися символами.

Естественно, реализации вычислительных моделей языков Scheme и Lisp существенно различаются. Прежде всего это относится к алгоритму вычисления S-выражения, а также к обработке «лексических контекстов». Тем не менее, существенное количество кода оказывается для реализации этих моделей общим.

При реализации общая часть кода класса `SchemeContinuation` была выделена в отдельный абстрактный базовый класс, получивший название `IntelibContinuation`. В этом классе описываются два виртуальных метода: `JustEvaluate`, призванный задавать общий алгоритм вычисления выражения (чисто виртуальный) и `CustomCommand` для реализации дополнительных кодов операций (в базовом классе пустой). Кроме того, в классе предусмотрены методы для работы с логическими значениями. Это позволило реализовать ряд библиотечных функций в одном экземпляре для обоих диалектов (так, функция `STRING?` для Scheme и функция `STRINGP` для диалекта Lisp имеют одну и ту же реализацию).

Реализация всех описанных выше кодов операций была вынесена в класс `IntelibContinuation`. В новой версии класса `SchemeContinuation`, наследуемой от `IntelibContinuation`, вводится одна дополнительная команда `case_check` для реализации формы `CASE`. В классе `LispContinuation`, предназначенном для диалекта Lisp, вводятся две дополнительные команды, а именно, `take_result_as_form` (вычисляет форму, находящуюся на вершине стека результатов; используется в реализации макросов) и `duplicate_last_result` (извлекает из стека результатов одно значение и помещает его обратно дважды; используется в реализации `SETQ` и `SETF`).

В общую часть реализации вычислительных моделей Lisp и Scheme были вынесены также базовые классы, представляющие понятия функции и спецформы (спецсинтаксиса в терминах Scheme). Реализация множества библиотечных функций оказалась не зависящей от диалекта; в качестве примера можно назвать функцию `COND`.

С другой стороны, для нужд диалекта Lisp пришлось выполнить реализацию динамического связывания переменных, реализовать концепцию обобщённого присваивания (`SETF`), сделать нерекурсивную реализацию функции `SORT`, аналогичной одноимённой функции из Common Lisp [10], которая предполагает передачу в качестве параметров, во-первых, функции, задающей отношение порядка и, во-вторых, функции-селектора.[3]

После реализации указанных возможностей для диалекта Lisp показалось естественным ввести соответствующие возможности и в Scheme. Таким образом, в настоящий момент InteLib Scheme является единственным диалектом языка Scheme, поддерживающим форму `SETF`; кроме того, поддерживается и функция `SORT`, во многих случаях весьма удобная. С другой стороны, в InteLib Lisp была введена поддержка `CALL/CC`.

6 Заключение

Полученный опыт позволяет сделать несколько неожиданное предположение о том, что вычислительные модели, основанные на рекурсии, следует реализовывать без использования рекурсии. Такое предположение нуждается, безусловно, в дальнейшей экспериментальной проверке и теоретическом обосновании.

Полученный в ходе работы диалект языка Scheme не соответствует существующим стандартам этого языка. С одной стороны, некоторые возможности в имеющемся диалекте не были реализованы за отсутствием достаточного времени (это касается прежде всего подсистемы макросов). С другой стороны, некоторые возможности, предусмотренные стандартом, плохо вписываются в чисто компилируемое окружение, определяемое использованием C++. В частности, к этой категории возможностей относится поиск символа по имени и извлечение

[3] Интересно заметить, что в пятой версии стандарта Scheme [9] функции `SORT` просто нет.

имени символа: действительно, имена переменных, роль которых в Scheme играют символы — это сущность, которая в чисто компилируемом коде должна полностью исчезать во время компиляции (например, результирующий исполняемый код не должен, по идее, изменяться при переименовании переменных в исходном коде).

В то же время, реализация, выполненная в рамках проекта InteLib, может быть использована и для синтеза встроенных интерпретаторов, и в этом случае стандартные возможности, обусловленные интерпретируемой сущностью Scheme, могут оказаться полезны.

Сказанное приводит нас к идее создания иерархии *уровней совместимости* реализуемого диалекта со стандартами в зависимости от конкретной ситуации. Выделение таких уровней и определение набора возможностей, доступных на каждом из них, представляется логичным направлением дальнейших исследований в области интеграции вычислительной модели языка Scheme в проекты на C++.

Список литературы

[1] E. Bolshakova and A. Stolyarov. Building functional techniques into an object-oriented system. In *Knowledge-Based Software Engineering. Proceedings of the 4th JCKBSE*, volume 62 of *Frontiers in Artificial Intelligence and Applications*, pages 101–106, Brno, Czech Republic, September 2000. IOS Press, Amsterdam.

[2] А. В. Столяров. Интеграция изобразительных средств альтернативных языков программирования в проекты на C++. Рукопись депонирована в ВИНИТИ 06.11.2001, №2319-В2001, Москва, 2001.

[3] И. Г. Головин, А. В. Столяров. Объектно-ориентированный подход к мультипарадигмальному программированию. *Вестник МГУ*, сер. 15 (ВМиК), №1, 2002 г., стр. 46–50.

[4] А. В. Столяров. Расширенный функциональный аналог языка Рефал для мультипарадигмального программирования. // Л. Н. Королев, ред., *Программные системы и инструменты. Тематический сборник*, том 2, стр. 184–195. Издательский отдел факультета ВМиК МГУ, Москва, 2001.

[5] А. Столяров, Е. Большакова, Н. Баева. InteLib Lisp в обучении программированию на Лиспе. // Тезисы докладов конференции «Свободное программное обеспечение в высшей школе», Переславль, 28–29 января 2006 года.

[6] Stolyarov, A. V. A framework of heterogenous dynamic data structures for object-oriented environment: the S-expression model In *Knowledge-Based Software Engineering. Proceedings of the 6th JCKBSE*, vol. 108 of *Frontiers in Artificial Intelligence and Applications*, pages 75–82, Protvino, Russia, August 2004. IOS Press.

[7] А. Столяров. Библиотека InteLib — инструмент мультипарадигмального программирования. // II конференция разработчиков свободных программ «На Протве». Тезисы докладов. Обнинск, 25–27 июля 2005 г., стр. 56–62.

[8] J. McCarthy. Recursive functions of symbolic expressions and their computation by machine. *Communications of the ACM*, 3(4):184–195, Apr 1960.

[9] R. Kelsey, W. Clinger, and J. Rees. Revised^5 report on Algorithmic Language Scheme. *ACM SIGPLAN Notices*, 33(9):26–76, 1998.

[10] G. L. Steele Jr. *Common Lisp the Language.* Digital Press, Burlington MA, second edition, 1990.

[11] Dan Piponi. Continuations in C.
`http://homepage.mac.com/sigfpe/Computing/continuations.html`

[12] J. Alger. *C++ for real programmers.* AP Professional, Boston, 1998. Русский перевод: *Джефф Элджер*, C++: библиотека программиста. СПб., Питер, 2001.

РЕФЕРАТЫ

К. Н. Долгова, В. Ю. Антонов. ВВЕДЕНИЕ В ЗАДАЧУ ДЕКОМПИЛЯЦИИ // СБОРНИК СТАТЕЙ МОЛОДЫХ УЧЕНЫХ факультета ВМиК МГУ, 2008, выпуск №5, стр. 5–15. В статье даётся краткое введение в проблематику задачи декомпиляции программ и рассматриваются возможности и недостатки существующих инструментальных средств для декомпиляции программ

Библиография 16 раб.

А. А. Фролов. АДАПТИВНЫЙ ПОИСК В ИНТЕЛЛЕКТУАЛЬНОЙ ОБУЧАЮЩЕЙ СИСТЕМЕ // СБОРНИК СТАТЕЙ МОЛОДЫХ УЧЕНЫХ факультета ВМиК МГУ, 2008, выпуск №5, стр. 16–23. Статья посвящена особенностям адаптивного поиска учебной информации в интеллектуальной обучающей системе и описанию архитектуры экспериментальной реализации. Также детально рассмотрен алгоритм поиска. Реализованная система позволяет осуществлять работу с различными учебными курсами и производить высокоуровневый поиск материала

Библиография 8 раб.

О. Г. Фролова. БИБЛИОТЕЧНАЯ РЕАЛИЗАЦИЯ ВЫЧИСЛИТЕЛЬНОЙ МОДЕЛИ ЯЗЫКА ПЛЭНЕР // СБОРНИК СТАТЕЙ МОЛОДЫХ УЧЕНЫХ факультета ВМиК МГУ, 2008, выпуск №5, стр. 24–33. В статье рассматривается возможность реализации вычислительной модели языка функционального программирования Плэнер в виде библиотеки классов языка C++. Такая реализация позволит программисту, использующему C++ в качестве основного языка в своих программах, воспользоваться возможностями, предоставляемыми языком Плэнер.

Библиография 10 раб.

А. В. Холмов. ОБОСНОВАНИЕ МЕТОДА РАЗУМНЫХ ЦЕЛЕЙ ДЛЯ ЗАДАЧ С НЕОПРЕДЕЛЕННОСТЬЮ // СБОРНИК СТАТЕЙ МОЛОДЫХ УЧЕНЫХ факультета ВМиК МГУ, 2008, выпуск №5, стр. 34–48. Данное исследование описывает и обосновывает методику поддержки принятия решений для задач многокритериальной оптимизации при наличии неопределенности. Приводится пример применения методики, где было показано, что сам факт учета неопределенности ведет к резкому росту числа отбираемых альтернатив, в то время как величина интервала неопределенности мало влияет на результат.

Библиография 8 раб.

А. П. Котляров. ИССЛЕДОВАНИЕ ЗНАЧЕНИЙ ПОКАЗАТЕЛЯ СЕМАНТИЧЕСКОЙ СОВМЕСТИМОСТИ ДЛЯ ПАР СЛОВ ПРИЛАГАТЕЛЬНОЕ–СУЩЕСТВИТЕЛЬНОЕ // СБОРНИК СТАТЕЙ МОЛОДЫХ УЧЕНЫХ факультета ВМиК МГУ, 2008, выпуск №5, стр. 49–53. Описываются результаты исследования значений показателя семантической совместимости, вычисляемого по частотам употребления слов, полученным от поисковой машины Яндекс. Для оценки берутся словосочетания вида прилагательное–существительное из словаря системы КроссЛексика и словарей фразеологизмов. Делаются выводы о применимости показателя для определения семантической допустимости словосочетаний и автоматизированного пополнения словаря КроссЛексики.

Библиография 8 раб.

И. А. Куров. СТРУКТУРНО НЕУСТОЙЧИВЫЕ ВЕКТОРНЫЕ КОНФИГУРАЦИИ В УРБАНИСТИКЕ // СБОРНИК СТАТЕЙ МОЛОДЫХ УЧЕНЫХ факультета ВМиК МГУ, 2008, выпуск №5, стр. 54–59. В этой статье представлены результаты математического моделирования развития различных урбанистических систем.

Библиография 3 раб.

Н. А. Малыш. ПРИНЦИП МАКСИМУМА ПОНТРЯГИНА ДЛЯ НЕАВТОНОМНЫХ МОНОТОННЫХ ЗАДАЧ ОПТИМАЛЬНОГО УПРАВЛЕНИЯ НА БЕСКОНЕЧНОМ ИНТЕРВАЛЕ ВРЕМЕНИ. // СБОРНИК СТАТЕЙ МОЛОДЫХ УЧЕНЫХ факультета ВМиК МГУ, 2008, выпуск №5, стр. 60–67. В статье получено необходимое условие оптимальности для неавтономных задач оптимального управления на бесконечном интервале времени, содержащее дополнительные условия на поведение гамильтониана задачи на бесконечности.

Библиография 10 раб.

А. С. Марков. О ВЛИЯНИИ СТЕПЕНИ СУММИРУЕМОСТИ КОЭФФИЦИЕНТОВ ДИФФЕРЕНЦИАЛЬНОГО ОПЕРАТОРА НА СКОРОСТЬ РАВНОСХОДИМОСТИ СПЕКТРАЛЬНЫХ РАЗЛОЖЕНИЙ. // СБОРНИК СТАТЕЙ МОЛОДЫХ УЧЕНЫХ факультета ВМиК МГУ, 2008, выпуск №5, стр. 68–72. В статье получены оценки скорости равносходимости с тригонометрическим рядом Фурье спектральных разложений широкого класса функций, имеющих особую асимптотику нормированных коэффициентов Фурье, на всем интервале и любом вложенном отрезке.

Библиография 3 раб.

А.А.Мартыненко. СТЕГАНОГРАФИЧЕСКИЙ АЛГОРИТМ ВНЕДРЕНИЯ ИНФОРМАЦИИ В ФАЙЛЫ ФОРМАТА BMP, УСТОЙЧИВЫЙ К СЖАТИЮ JPEG // СБОРНИК СТАТЕЙ МОЛОДЫХ УЧЕНЫХ факультета ВМиК МГУ, 2008, выпуск №5, стр. 73–75. В статье описан алгоритм скрытого внедрения служебной информации в файлы формата BMP

Библиография 4 раб.

О. В. Матусевич. СРАВНЕНИЕ КОНСЕРВАТИВНЫХ СХЕМ И СХЕМ СУММАРНОЙ АППРОКСИМАЦИИ ДЛЯ УРАВНЕНИЯ ШРЕДИНГЕРА В АКСИАЛЬНО-СИММЕТРИЧНОЙ СРЕДЕ // СБОРНИК СТАТЕЙ МОЛОДЫХ УЧЕНЫХ факультета ВМиК МГУ, 2008, выпуск №5, стр. 76–82. В статье приводится сравнение методов суммарной аппроксимации и консервативных схем для задачи распространения оптического излучения в аксиально-симметричной нелинейной среде. Показано, что использование схем, не сохраняющих разностные аналоги инвариантов задачи, может приводить к существенным искажениям решения.

Библиография 16 раб.

А. Д. Поспелов. О СЛОЖНОСТИ УМНОЖЕНИЯ ПОЛИНОМОВ И МАТРИЦ // СБОРНИК СТАТЕЙ МОЛОДЫХ УЧЕНЫХ факультета ВМиК МГУ, 2008, выпуск №5, стр. 83–97. Статья посвящена построению эффективных алгоритмов умножения полиномов многих переменных и оценкам сложности умножения матриц путем сведения этих задач к умножению в групповых алгебрах.

Библиография 28 раб.

А. И. Пучкова. О СТАБИЛИЗАЦИИ ЦЕПОЧКИ МНОЖЕСТВ ПРИ ПОСТРОЕНИИ МИНИМАЛЬНОЙ ВЫПУКЛОЙ ОБОЛОЧКИ // СБОРНИК СТАТЕЙ МОЛОДЫХ УЧЕНЫХ факультета ВМиК МГУ, 2008, выпуск №5, стр. 98–100. В статье рассматривается вопрос о стабилизации цепочки множеств, которая используется для построения минимальной выпуклой оболочки множества. Приводится точная оценка сверху для номера, начиная с которого наступает стабилизация.

Библиография 4 раб.

И. Г. Шевцова. ОБ АБСОЛЮТНОЙ ПОСТОЯННОЙ В НЕРАВЕНСТВЕ БЕРРИ–ЭССЕЕНА // СБОРНИК СТАТЕЙ МОЛОДЫХ УЧЕНЫХ факультета ВМиК МГУ, 2008, выпуск №5, стр. 101–110. Верхняя оценка абсолютной постоянной в классическом неравенстве Берри–Эссеена для сумм независимых одинаково распределённых случайных величин снижена до 0.7005. Также указан метод дальнейшего уточнения верхней оценки.

Ключевые слова и фразы: центральная предельная теорема, нормальная аппроксимация, оценка скорости сходимости, сумма независимых случайных величин, неравенство Берри–Эссеена, дробь Ляпунова.

Библиография 23 раб.

К. А. Симонян, С. В. Гришин, Д. С. Ватолин. АДАПТИВНЫЙ МЕТОД ОЦЕНКИ ДВИЖЕНИЯ В ВИДЕО // СБОРНИК СТАТЕЙ МОЛОДЫХ УЧЕНЫХ факультета ВМиК МГУ, 2008, выпуск №5, стр. 111–118. В статье описан адаптивный метод оценки движения, нацеленный на обнаружение истинного движения в видеопоследовательности. Приведены результаты сравнения описанного метода с другими для различных сценариев обработки видео.

Библиография 5 раб.

А. В. Столяров. ИМПОРТ ВЫЧИСЛИТЕЛЬНОЙ МОДЕЛИ ЯЗЫКА SCHEME В ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ОКРУЖЕНИЕ // СБОРНИК СТАТЕЙ МОЛОДЫХ УЧЕНЫХ факультета ВМиК МГУ, 2008, выпуск №5, стр. 119–130. В статье рассматривается объектно-ориентированная реализация вычислительной модели языка Scheme в виде библиотеки классов C++, выполненная в рамках проекта InteLib и предназначенная для мультипарадигмального программирования. Основное внимание уделяется реализации механизма *продолжений* (continuations).

Библиография 10 раб.
