ПРОГРАММНЫЕ СРЕДСТВА И МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

Серия

“КОНСТРУИРОВАНИЕ И ОПТИМИЗАЦИЯ ПРОГРАММ”

Под редакцией
доктора физ.-мат. наук, профессора, чл.-корр. РАЕН
В. Н. Касьянова

Выпуски серии:

1. Смешанные вычисления и преобразование программ (1991)
2. Конструирование и оптимизация программ (1993)
3. Интеллектуализация и качество программного обеспечения (1994)
4. Проблемы конструирования эффективных и надежных программ (1995)
5. Оптимизирующая трансляция и конструирование программ (1997)
6. Проблемы систем информатики и программирования (1999)
7. Поддержка супервычислений и Интернет-ориентированные технологии (2001)
8. Касьянов В. Н., Мирзуитова И. Л. Slicing: срезы программ и их использование (2002)
9. Современные проблемы конструирования программ (2002)
10. Новые информационные технологии в науке и образовании (2003)
11. ***Программные средства и математические основы информатики***

Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

ПРОГРАММНЫЕ СРЕДСТВА И МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

Под редакцией
проф. Виктора Николаевича Касьянова

Новосибирск 2004

УДК 519.68; 681.3.06
ББК З 22.183.49+ З 22.174.2

Программные средства и математические основы информатики. — Новосибирск: Ин-т систем информатики им. А. П. Ершова СО РАН, 2004. — 278 с.

Является одиннадцатым в серии сборников, издаваемых Институтом систем информатики им. А.П.Ершова СО РАН по проблемам конструирования и оптимизации программ. Посвящен решению актуальных задач, связанных с программными средствами и математическими основами информатики.

Сборник представляет интерес для системных программистов, а также студентов и аспирантов, специализирующихся в области системного и теоретического программирования.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

SOFTWARE TOOLS AND MATHEMATICAL FOUNDATIONS OF INFORMATICS

Edited by
prof. V. N. Kasyanov

Novosibirsk 2004

This volume is the eleventh one in a series of books published in A.P. Ershov Institute of Informatics Systems on the problems of program construction and optimization. This volume is devoted to a decision of actual problems connected with software tools and mathematical foundations of informatics.

The volume is of interest for system programmers, students and postgraduates working in the field of system and theoretical programming.

ПРЕДИСЛОВИЕ РЕДАКТОРА

Одиннадцатый выпуск серии «Конструирование и оптимизация программ» посвящен решению актуальных задач, связанных с теоретическими основами информатики и созданием программных систем.

Продолжая уже сложившиеся традиции, данный выпуск, как и предыдущие, базируется на результатах исследований, выполненных в лаборатории по конструированию и оптимизации программ Института систем информатики СО РАН совместно с Новосибирским государственным университетом при финансовой поддержке Российского фонда фундаментальных исследований, Российского гуманитарного научного фонда, Минобразования Российской Федерации, а также компании Майкрософт.

Открывает сборник статья Т.А. Волянской, посвященная международным стандартам представления в сети Интернет информационных ресурсов по культурному наследию. В ней содержится краткий обзор стандарта ANSI/NISO Z39.50 и профиля CIMI. Стандарт Z39.50 определяет прикладную службу и спецификацию протокола для поиска и извлечения информации из баз данных. Профиль CIMI служит спецификацией использования стандарта Z39.50 для доступа к информации о культурном наследии.

Статья П.А. Дортмана посвящена оптимизации программ в системе параллельного программирования SFP на базе функционального языка программирования SISAL 3.0. В ней описаны алгоритмы для выполнения некоторых традиционных оптимизирующих преобразований над SISAL-программами, представленными в виде так называемых потоковых графов.

В статье А.А. Дунаева исследуются способы организации работы с большими линейными массивами данных. В ней рассмотрены несколько стратегий предварительной выборки данных и описаны методы обработки нуклеотидных последовательностей, а также способы отображения результатов вычислений.

Понятие NUMA-архитектуры возникло в конце 80-х годов с появлением интереса к компьютерам с распределенной памятью. В статье В.А. Евстигнеева излагаются основы теории полиномов Эрхарта и её применения к некоторым проблемам, возникающим в NUMA-архитектурах.

В последнее время адаптивные гипермедиа-системы становятся все более популярными в дистанционном обучении, предоставляя средства доступа к информации, управляемые пользователем. Адаптивные гипермедиа-системы сводят воедино идеи гипермедиа-систем и интеллектуальных обучающих систем и делают возможным персонализированный доступ к ин-

формации. В статье В.Н. Касьянова и Е.В. Касьяновой рассматриваются вопросы поддержки дистанционного обучения. Особое внимание авторы уделяют анализу методов и средств адаптивной гипермедиа, используемых современными адаптивными обучающими Web-системами.

Статья В.Н. Касьянова и И.Л. Мирзуитовой посвящена алгоритмам распараллеливания циклов — одного из наиболее эффективных реструктурирующих преобразований, используемых в распараллеливающих компиляторах. Авторы приводят описание основных алгоритмов распараллеливания циклов и дают сопоставление их сильных и слабых сторон как на примерах, так и в сравнении по получению «оптимальных» результатов.

В статье Е.В. Касьяновой кратко представлен новый язык программирования Zonnon для платформы .NET, работа над которым ведется в Цюриховском институте информатики. Разрабатываемый язык задуман авторами как современная альтернатива хорошо известному языку Оберон, являющемуся преемником языков Паскаль и Модула-2.

Новые подходы и технологии могут существенно изменить процесс взаимодействия людей при коллективной работе. Статья Ю.В. Малининой описывает опыт внедрения электронной среды для организации совместной работы по накоплению и каталогизации информации по преобразованиям программ на основе технологии WikiWiki.

Статья В.А. Маркина и С.А. Маркиной посвящена системе ПРОГРЕСС-2, которая создается как конструктор для построения прототипа компилятора, кирпичиками которого являются различные функциональные и инструментальные компоненты. Особое внимание в статье уделяется ядру системы и средствам задания сценария работы создаваемого компилятора.

В статье А.Л. Серебренникова рассматриваются задачи, решаемые нейросетями, описываются методы обучения нейросетей и сравнительные тесты методов, а также направления и подходы в развитии интерфейсной части разрабатываемой автором среды Significo, предназначенной для создания, обучения и диагностики нейросетей.

Завершает сборник статья Е.С. Черемушкина, Т.Г. Коноваловой, Ф.А. Мурзина и А.Э. Кель, в которой представлен разработанный авторами инструментарий, позволяющий производить полноценный поиск цис-элементов на последовательностях ДНК, наиболее полно использующий данные, имеющиеся у экспериментатора.

Проф. В.Н. Касьянов

Т. А. Волянская

МЕЖДУНАРОДНЫЕ СТАНДАРТЫ ПРЕДСТАВЛЕНИЯ В СЕТИ ИНТЕРНЕТ ИНФОРМАЦИОННЫХ РЕСУРСОВ ПО КУЛЬТУРНОМУ НАСЛЕДИЮ: СТАНДАРТ ANSI/NISO Z39.50 И ПРОФИЛЬ CIMI*

ВВЕДЕНИЕ

В настоящее время в сети Интернет представлено большое число разнородных и обособленных информационных ресурсов по культурному наследию. В их число входят различные электронные музеи, архивы и каталоги. Информационные ресурсы по культурному наследию имеют самый разнообразный характер: это описательные базы данных, базы данных изображений, видео- и аудио- материалов. Эти ресурсы принадлежат различным организациям, которые проводят самостоятельную политику в отношении их описания, использования и публичного доступа к ним.

Предоставление доступа к информации о культурном наследии организовано различными способами и с применением различных технологий и протоколов, которые часто не согласуются с существующими стандартами. Наиболее распространенным способом является использование Web-технологии с доступом по HTTP протоколу, одним важным недостатком которого является отсутствие глобальной стандартизации на уровне организации данных и форматов их представления. Большинство существующих информационно-поисковых систем поддерживают совершенно различные структуры хранения данных, способы доступа и форматы представления информации и, как следствие, имеют свой собственный пользовательский интерфейс.

Одна из важных задач в области разработки распределенных информационно-поисковых систем по культурному наследию — это унификация доступа и интеграция информационных ресурсов [2]. Целью является объединение имеющихся информационных ресурсов по культурному наследию в единую распределенную информационную систему со сквозным поиском. Это позволит открыть для пользователей всю информацию, накопленную за

* Работа выполнена при финансовой поддержке Российского гуманитарного научного фонда (грант № 02-05-12010).

долгое время и в разных местах, и оперативно получать исчерпывающие ответы на сложные запросы. Однако, для разнородных и обособленных информационных систем, не поддерживающих существующие стандарты, возникает проблема с унификацией доступа к данным и интеграцией информационных ресурсов. Все это во многом затрудняет как работу с ними, так и дальнейшее использование предоставляемой ими информации.

Единственная технология в настоящее время, которая позволяет решить вышеперечисленные проблемы, — это технология, основанная на международных стандартах ANSI/NISO Z39.50 (ISO 23950) [1,5,6] и профиле CIMI [4,10]. Стандарт Z39.50 позволяет унифицировать сетевой доступ к любым базам данных, а профиль CIMI регламентирует доступ к информации о культурном наследии по протоколу Z39.50. Информационные системы по культурному наследию, организованные на основе Z39.50 серверов с использованием CIMI профиля, становятся независимыми от конкретных систем хранения данных и, следовательно, могут быть интегрированы с другими подобными системами.

Технологию, основанную на стандарте Z39.50 и профиле CIMI, предполагается использовать при создании виртуального музея истории информатики в Сибири, работа над которым ведется коллективом сотрудников ИСИ СО РАН, ИМ СО РАН и НГУ при финансовой поддержке Российского гуманитарного научного фонда (грант РГНФ N 02-05-12010) [3].

Статья содержит краткий обзор стандарта Z39.50 и профиля CIMI. Разд. 1 посвящен истории разработки стандарта Z39.50 и профиля CIMI. В разд. 2 рассматриваются основы стандарта Z39.50, модель данных и обеспечиваемые сервисные возможности. В разд. 3 кратко изложена модель поиска информации по Z39.50, рассматриваются понятия абстрактной базы данных и пунктов доступа, приводятся наборы и типы поисковых атрибутов, обсуждаются построение RPN запросов и результаты поиска. В разд. 5 приведена модель извлечения информации по Z39.50, рассматриваются понятия схемы базы данных, наборов тэгов, абстрактной структуры записи, на примере иллюстрируются введенные понятия. Пятый раздел статьи посвящен краткому обзору профиля CIMI, рассматриваются следующие основные аспекты профиля: поиск записей и набор атрибутов CIMI-1; выбор записей и связанные с ним набор тэгов CIMI Tag Set, CIMI схема и абстрактная структура записи с различными уровнями семантической интероперабельности, а также передача записей и синтаксис записи GRS-1. В разд. 6 приведен пример распределенной информационной системы на базе Z39.50.

1. ИСТОРИЯ РАЗРАБОТКИ Z39.50 И CIMI

Стандарт Z39.50 (Information Retrieval (Z39.50): Application Service Definition and Protocol Specification) определяет прикладную службу и спецификацию протокола для поиска и извлечения информации из баз данных. Он предназначен для унификации сетевого доступа к базам данных и определяет процедуры поиска, извлечения и форматы представления информации [1,5,6].

Первая версия протокола, получившего название Z39.50, была подготовлена Комитетом организации по национальным информационным стандартам США — NISO (National Information Standards Organization) и введена в 1988 г. стандартом Z39.50-1988. Действие этой версии распространялось только на работу с библиографической информацией.

В 1989 г. было организовано Агентство поддержки протокола Z39.50 (Maintenance Agency Z39.50) под административным управлением лаборатории Конгресса США, а в 1990 г. сформирована Группа исполнителей Z39.50 (Z39.50 Implementors Group — ZIG). Ее членами стали производители, продавцы, распространители разнородных видов информации (библиографической, текстовой, графической, финансовой, химической и т.д.). Агентство поддержки Z39.50 является постоянно действующим органом, которое занимается сопровождением и развитием этого стандарта. Сетевой адрес агентства — http://lcweb.loc.gov/z3950/agency, здесь находятся вся информация по протоколу, новости, документация, реестры объектов и т.п.

В 1992 г. указанными организациями была разработана версия 2 стандарта (Z39.50-1992), заменившая стандарт 1988 г. Версия 3 стандарта (Z39.50-1995) была разработана в 1995 г. [5]. Поскольку стандарт Z39.50-1995 является развитием версии 1992 г. и совместим с ней, он определяет протокол версий 2 и 3. В 1995 г. протокол Z39.50 был принят как американский национальный стандарт ANSI (ANSI/NISO Z39.50-1995), а в ноябре 1998 г. — как международный стандарт ISO-23950.

Стандарт ANSI/NISO Z39.50-2003 [6], действующий в настоящее время, был утвержден в ноябре 2002 г. Он является технической переработкой стандарта ANSI/NISO Z39.50-1995 и также определяет версии 2 и 3, но дополнительно включает различные пояснения, исправления и соглашения, рекомендованные группой исполнителей Z39.50 — ZIG.

В первые годы своего существования протокол использовался преимущественно для организации доступа к библиографическим ресурсам, на сегодняшний день область его применения существенно расширена. Он используется для доступа к научно-технической, финансовой информации,

к геоинформационным ресурсам, к глобальным базам метаданных, тезаурусам и рубрикаторам, цифровым коллекциям [9] и музейной информации [10].

Профиль CIMI, предназначенный для работы с информацией о культурном наследии по протоколу Z39.50, был разработан в 1998 г. Консорциумом по компьютерному обмену музейной информацией — CIMI (Consortium for the Computer Interchange of Museum Information) [10]. Это некоммерческая инициатива, занимающаяся развитием коммуникативных стандартов, сохранением и обменом музейной информацией в электронном виде. Сетевой адрес консорциума — http://www.cimi.org. При создании профиля рабочая группа CIMI Z39.50 объединила вместе экспертов по Z39.50, экспертов в области музейного дела и музейной информации, разработчиков программного обеспечения и специалистов в области коммерции.

Профиль CIMI служит спецификацией использования Z39.50 для поиска и извлечения информации о культурном наследии. Он определяет подмножество характеристик, опций и параметров Z39.50, необходимых для поддержки функциональности и требований пользователя при поиске и извлечении информации о культурном наследии. Элементы этого профиля имеют глобальные идентификаторы и являются частью международного стандарта ISO-23950 [5,6].

2. ОСНОВЫ СТАНДАРТА Z39.50

Стандарт Z39.50 определяет службу и протокол типа клиент/сервер для информационного поиска. Он специфицирует процедуры и форматы для проведения клиентом поиска в базах данных, предоставляемых сервером, извлечения записей из баз данных и выполнения других функций, связанных с информационным поиском. Протокол Z39.50 определяет взаимодействие только между клиентскими и серверными информационно-поисковыми приложениями, он не определяет взаимодействие между клиентом и конечным пользователем. Точнее говоря, протокол Z39.50 не определяет форматы хранения данных в конкретных базах данных, способы их индексации и процедуры функционирования различных СУБД. Он также не определяет интерфейсы взаимодействия пользователя и клиента.

Не вдаваясь в детали работы протокола, можно сказать, что стандарт Z39.50 определяет такие правила взаимодействия компьютеров, которые позволяют унифицировать доступ к различным базам данных. Таким образом, пользователь, использующий всего лишь одно клиентское приложение,

может производить поиск информации в удаленных распределенных базах данных, имеющих самую различную структуру и форматы представления информации.

Две основные особенности отличают протокол Z39.50 от других протоколов. Во-первых, это *абстрактная модель представления информации*. В идеологии Z39.50 в рамках одной схемы данных все базы данных совершенно одинаковы, несмотря на их физические различия по используемой СУБД, полям и синтаксису запросов. Иными словами, протокол предоставляет абстрактную модель представления информации на каждом этапе взаимодействия клиента и сервера. В Z39.50 клиент работает всегда с одной и той же системой запросов и получает данные в одних и тех же форматах.

Вторая особенность состоит в том, что протокол Z39.50 полностью обеспечивает *сессионное взаимодействие* клиента и сервера. Эта особенность заложена в самом протоколе и реализуется во всех его приложениях, будь то серверная система или программа-клиент.

Z39.50 обеспечивает следующие основные **сервисные возможности**.

- Клиент может производить поиск, указав список серверов и список баз данных для поиска, используя параметры, определяющие записи, идентифицируемые для поиска.
- Сервер отвечает на запрос клиента подсчетом найденных записей (всех или их части).
- Клиент может получить у сервера отобранные записи. При этом клиент принимает условие, что записи, отбираемые при поиске, формируются и определяются установками сервера.
- Клиент может определить установки для получения данных в том случае, когда он не уверен в результатах поиска: в зависимости от количества найденных записей производить их передачу в полном объеме или в сокращенной форме.
- Клиент может установить предпочтительную для него форму (синтаксис) получения данных, например, в USMARC'е.
- Клиент может присвоить имя полученному набору данных для последующего сравнения результатов, а впоследствии удалить названный им результат поиска.
- Сервер может налагать ограничения контроля доступа на клиента, которые определяются результатами аутентификации, предшествующей обслуживанию запроса.
- Сервер может производить контроль ресурсов для посылки незапрашиваемых или запрашиваемых сообщений. При этом сервер может временно приостанавливать обслуживание клиента и пре-

доставлять ему возможность для определения целесообразности продолжения работы.

Цель этого стандарта — облегчить взаимодействие между клиентами и серверами в прикладных системах, в которых клиент ведет поиск и извлекает записи из баз данных сервера. Базы данных могут иметь различную реализацию: разные системы могут иметь разные способы хранения данных и разные способы доступа к ним. Поэтому при описании баз данных в Z39.50 используется общая *абстрактная модель базы данных*, которой каждая отдельная система может поставить в соответствие свою реализацию. Это позволяет разным системам при взаимодействии использовать стандартные и общепонятные термины для задач поиска и извлечения информации из баз данных. Модели поиска и извлечения информации будут рассмотрены позже в третьем и четвертом разделах соответственно.

Термин *база данных*, использующийся в данном стандарте, обозначает собрание записей, где каждая запись представляет собой собрание взаимосвязанных данных. Термин *запись базы данных* относится к локальной структуре данных, представляющей информацию отдельной записи. С базой данных связаны одно или более множеств *пунктов доступа*, которые могут быть указаны при поиске записей в базе данных, и одно или более множеств *элементов*, которые могут быть извлечены из записи базы данных. *Пункт доступа* — это уникальный или неуникальный ключ, который может быть указан отдельно или в сочетании с другими пунктами доступа при поиске записей. Пункт доступа может, но не обязан быть связанным с некоторым элементом. Он может быть эквивалентен элементу, может быть производным от некоторого множества элементов или может не быть связанным ни с какими элементами.

Для наглядности рассмотрим следующий простой пример: пусть у нас есть база данных, в которой содержатся данные о музейных предметах. Информация классифицируется по следующим полям: ID, Title, Type, Creator_name, Date_of_origin, Place_of_origin, Material, Technique, Dimensions, Period, Keywords, Description.

Таблица 1

Список полей базы данных музейных предметов

Название поля	**Комментарий**
ID	Идентификационный номер предмета
Title	Название предмета
Type	Типология
Creator_name	Создатель
Date_of_origin	Дата создания
Place_of_origin	Место создания
Material	Материал
Technique	Техника
Dimensions	Размеры, вес
Period	Культурный период
Keywords	Ключевые слова
Description	Краткое описание предмета

С этой базой данных можно связать, например, следующее множество пунктов доступа, которое может быть указано при поиске записей (в скобках приведены соответствующие числовые значения поисковых Use атрибутов из набора атрибутов CIMI-1).

Таблица 2

**Множество пунктов доступа,
связанное с базой данных музейных предметов**

Название поля	**Пункты доступа**
ID	ObjectID (2024)
Title	Title (4), ObjectTitle (2033), What (2047)
Type	Material type (1031), ObjectName (2032)
Creator	Author (1003), CreatorName (2035), CreatorGeneral (2041), Who (2046)
DateOfOrigin	DateOfOrigin (2022), When (2048)
PlaceOfOrigin	Name geographic (58), PlaceOfOrigin (2023), Where (2049)
Material	MaterialMedium (2008)
Technique	ProcessTechnique (2012)
Dimensions	Dimensions (2074)
Period	StylePeriod (2017)
Keywords	Any (1016), SubjectContent (2040)
Description	Abstract (62)

Также с этой базой данных можно связать множество элементов, представленных в табл. 3, которые могут быть извлечены из записи, представленной в табл. 3.

Таблица 3

Множество элементов, связанное с базой данных музейных предметов

Название поля	**Элементы**
ID	Identifier (2,28), ObjectID (5,3)
Title	Title (2,1), ObjectTitle (5,32)
Type	Type (2,22), ObjectName (5,31)
Creator	Creator (2,2), CreatorInfo (5,36), CreatorGeneral (5,49)
DateOfOrigin	Date (2,8), DateOfOrigin (5,45)
PlaceOfOrigin	PlaceOfOrigin (5,11)
Material	MaterialMedium (5,5)
Technique	ProcessTechnique (5,12)
Dimensions	Dimensions (5,13)
Period	StylePeriod (5,14)
Keywords	Subject (2,21), Subject (5,2)
Description	Description (2,17)

В скобках приведены соответствующие числовые значения элементов из наборов тэгов Generic Tag Set и CIMI Tag Set. Эти наборы тэгов будут приведены позже в таблицах 14 и 19 соответственно. Первая цифра в скобках означает номер набора тэгов (2 для Generic, 5 для CIMI), а вторая — номер тэга в наборе.

3. МОДЕЛЬ ПОИСКА ИНФОРМАЦИИ

Для проведения поиска записей требуется указать список серверов баз данных и список имен баз данных для поиска, а также сформулировать поисковый запрос, содержащий критерии поиска. В традиционных системах, не использующих Z39.50, для построения запроса необходимо также знать дополнительную информацию: синтаксис языка запросов для каждой СУБД, а также структуру, имена полей и типы данных каждой базы данных. В случае, когда в группе выбранных серверов и баз данных перечисленные характеристики различны, даже очень простой запрос не сможет быть ис-

полнен для всей группы. В Z39.50 эта проблема решается за счет построения специфической модели поиска и стандартизации ее компонент.

В Z39.50 поисковые запросы всегда формулируются не к реальной базе данных, а к некой абстрактной. Эта *абстрактная база данных* не имеет никакой структуры, она характеризуется только *пунктами доступа* (поисковыми атрибутами). При получении от клиента запроса в виде термов и поисковых атрибутов, сервер конвертирует его в синтаксис реальной базы данных, и эта процедура остается незаметной для клиента. При таком подходе к процедуре поиска все базы данных становятся для клиента одинаковыми, если они поддерживают один и тот же набор пунктов доступа.

Наборы поисковых атрибутов составляют класс объектов Z39.50 {Z39.50 3}, подлежащих стандартизации. На настоящий момент стандартизованы наборы атрибутов, представленные в табл. 4.

Таблица 4

Наборы атрибутов Z39.50

OID	Набор	Комментарий
{Z39.50 3 1}	bib-1	Библиографическая информация
{Z39.50 3 2}	exp-1	Explain
{Z39.50 3 3}	ext-1	Расширенный сервис
{Z39.50 3 4}	ccl-1	Type-2 (ISO 8777) и Type-100 (Z39.58) запросы
{Z39.50 3 5}	Gils	Для поиска GILS
{Z39.50 3 6}	Stas	Научно-техническая информация
{Z39.50 3 7}	collections-1	Навигация по электронным коллекциям
{Z39.50 3 8}	cimi-1	Информация по музейным коллекциям
{Z39.50 3 9}	geo-1	Пространственные метаданные
{Z39.50 3 10}	ZBIG	Биологическая информация
{Z39.50 3 11}	Util	Утилиты
{Z39.50 3 12}	xd-1	Междоменный набор
{Z39.50 3 13}	Zthes	Навигация по тезаурусам

Набор атрибутов Bib-1 [7], используемый для поиска библиографической информации, является основным, его подмножества включены во многие другие наборы.

Типы поисковых атрибутов

Набор Bib-1 включает в себя шесть типов атрибутов с номерами 1-6: Use, Relation, Position, Structure, Truncation и Completeness. При построении

запроса указание поисковых атрибутов в комбинации с поисковым термом определяет критерии поиска. В каждой группе каждый атрибут определяется уникальным числовым значением, поэтому для указания поискового атрибута необходимо задание двух чисел: тип + значение.

Атрибуты 1: Use (атрибуты использования)

Атрибуты этого типа указывают, с какой смысловой информацией связывается поисковый терм, т.е. определяют пункты доступа. В наборе атрибутов Bib-1 определено 99 значений. Среди значений Use есть значения, соответствующие автору, заглавию, ключевым словам, году издания и т.п. Среди атрибутов Use есть такие, которые связываются одновременно с несколькими полями («Author-name-and-title»), а значение «Anywhere» связывается сразу со всеми поисковыми полями.

Таблица 5

Use атрибуты Bib-1

Значение	**Атрибут**	**Комментарий**
4	Title	Название
21	Subject	Ключевое слово
31	Date-publication	Дата публикации
1003	Author	Автор
1000	Author-name-and-title	Автор и название
1035	Anywhere	Везде

Атрибуты 2: Relation (атрибуты отношения)

Атрибуты *Relation* описывают отношения между пунктами доступа и поисковыми термами, т.е. они указывают, как поисковый терм соотносится с выбираемыми данными из полей, определенных атрибутом Use.

Таблица 6

Relation атрибуты Bib-1

Значение	Отношение	Комментарий
1	Less than	Меньше
2	Less than or equal	Меньше или равно
3	Equal	Равно
4	Greater or equal	Больше или равно
5	Greater than	Больше
6	Not equal	Не равно
100	Phonetic	Фонетические отношения
101	Stem	Корень
102	Relevance	Релевантность (соответствие)
103	AlwaysMatches	Любое соответствие

Атрибуты 3: Position (атрибуты позиции)

Атрибуты *Position* определяют позицию поискового терма внутри поля или подполя, т.е. указывают, в каком месте поля, определенного атрибутом Use, должен находиться поисковый терм.

Таблица 7

Position атрибуты Bib-1

Значение	Позиция	Комментарий
1	First in field	Первая позиция в поле
2	First in subfield	Первая позиция в подполе
3	Any position in field	Любая позиция в поле

Атрибуты 4: Structure (атрибуты структуры)

Атрибуты *Structure* указывают, какую структуру имеет поисковый терм.

Таблица 8

Structure атрибуты Bib-1

Значение	Структура	Комментарий
1	Phrase	Фраза
2	Word	Слово
3	Key	Ключ
4	Year	Год
5	Date (normalized)	Дата (нормализована)
6	Word list	Список слов

100	Date (un-normalized)	Дата (не нормализована)
101	Name (normalized)	Имя (нормализовано)
102	Name (un-normalized)	Имя (не нормализовано)
103	Structure	Структура
104	Urx	Идентификатор документа
105	Free-form-text	Текст в свободной форме
106	Document-text	Текст документа
107	Local-number	Локальный номер
108	String	Строка
109	Numeric string	Числовая строка

Атрибуты 5: Truncation (атрибуты усечения)

Атрибуты *Truncation* указывают, может ли один или несколько символов, находящихся в позиции, определяемой атрибутами Truncation, не учитываться при сопоставлении с поисковым термом.

Таблица 9

Truncation атрибуты Bib-1

Значение	**Усечение**	**Комментарий**
1	Right truncation	Усечение справа
2	Left truncation	Усечение слева
3	Left and right truncation	Слева и справа
100	Do not truncate	Без усечения
101	Process # in search term	Обработать # в каждом термине
102	RegExpr-1	Регулярные выражения 1
103	RegExpr-2	Регулярные выражения 2

Атрибуты 6: Completeness (атрибуты полноты)

Атрибуты *Completeness* указывают, что является обязательной областью совпадения при поиске, т.е. представляет ли поисковый терм полное, неполное подполе или поле, или, другими словами, могут ли дополнительные слова присутствовать вместе с поисковым термом в поле или подполе.

Таблица 10

Completeness атрибуты Bib-1

Значение	Полнота	Комментарий
1	Incomplete subfield	Неполное подполе
2	Complete subfield	Полное подполе
3	Complete field	Полное поле

Поисковые запросы

В Z39.50 предусмотрено несколько видов запросов: type-0 — ANY, type-1 — IMPLICIT RPNQuery, type-2 — OCTET STRING (CCL ISO 8777), type-100 — OCTET STRING (CCL Z39.58), type-101 — IMPLICIT RPNQuery, type-102 — OCTET STRING, type-104 — IMPLICIT EXTERNAL (SQL).

Запрос *type-0* представляет собой любой запрос в синтаксисе СУБД, с которой связан сервер, он передается без изменения. Запрос *type-104 (SQL)* — это новый тип запросов, которые входят в стандарт Z39.50 с 2000 г. На данный момент практически нет серверов, которые поддерживали бы SQL в Z39.50. Как видно, SQL-запрос — простая текстовая строка. Наибольший интерес представляют запросы *type-1* и *type-101* — запросы RPN, не различающиеся в 3 версии протокола Z39.50. Запросы *type-1* являются обязательными для всех серверов Z39.50, поддержка других типов запросов сервером Z39.50 является факультативной.

RPN-запрос — поисковый запрос, представленный в формате обратной польской нотации (RPN — reverse polish notation). *RPN-запрос* состоит из одного или нескольких условий на пункты доступа, связанных логическими операторами. Каждое условие на пункт доступа состоит из поискового терма и атрибутов. Структуру RPN-запроса можно представить в виде дерева, в узлах которого находятся связывающие логические операторы (AND, OR, AND-NOT), а листья — блоки «атрибуты+терм» (APT).

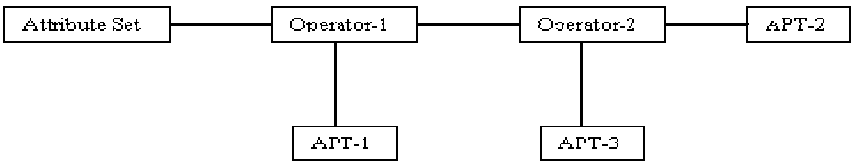


Рис. 1. Структура RPN-запроса

Например, для поиска музейных предметов, создателем которых является Иванов или Иванова, необходимо задать:

```
(CIMI-1 1=2035 2=3 3=3 4=2 5=1 6=2 «Иванов»)
```

Каждый атрибут — это пара, отражающая тип атрибута и его значение. В приведенном примере присутствуют следующие поисковые атрибуты: тип «Use» и значение «CreatorName» (1=2035), тип «Relation» и значение «Equal» (2=3), тип «Position» и значение «Any position in field» (3=3), тип «Structure» и значение «Word» (4=2), тип «Truncation» и значение «Right truncation» (5=1) и тип «Completeness» и значение «Incomplete subfield» (6=2).

Результаты поиска

В результате выполнения поиска в базах данных клиент может получить от сервера следующую информацию: сообщение об ошибке, количество найденных записей или сами найденные записи. Первый вариант ответа связан с какой-либо ошибкой, второй и третий варианты соответствуют успешному проведению поиска. Какой из них будет получен клиентом, зависит от параметров, которые передаются серверу вместе с поисковым запросом.

Для этого вводятся понятия *малый*, *средний* и *большой наборы*. Здесь под *набором* понимается совокупность найденных записей, пронумерованная сквозным образом. Все записи из малого набора всегда возвращаются, все записи из большого набора никогда не возвращаются, а из среднего набора возвращаются некоторые записи. Далее задаются следующие параметры.

- Верхняя граница малого набора, т.е. максимальный номер записи в малом наборе, который начинается с первой записи.
- Нижняя граница большого набора, т.е. номер, начиная с которого записи попадают в большой набор. Все записи, номера которых больше верхней границы малого набора, но меньше нижней границы большого, считаются записями из среднего набора.
- Количество возвращаемых записей из среднего набора.

Меняя эти три параметра, можно добиться возвращения любого количества записей, в том числе ни одной.

Наконец, следует отметить, что все найденные при поиске записи сервер должен сохранить в сессионном блоке для последующего использования. Если сервер допускает опцию присвоения имени результату поиска, этой

сохраненной совокупности может быть присвоено имя, если нет, совокупность сохраняется неименованной и переписывается при последующем поиске. Именованные результирующие наборы, которые сохраняются на сервере, можно использовать в последующих RPN-запросах, где они выступают такими же операндами, как блоки APT.

4. МОДЕЛЬ ИЗВЛЕЧЕНИЯ ИНФОРМАЦИИ

После того как произведен поиск и создан результирующий набор на сервере, записи из результирующего набора можно извлекать и просматривать. Для того чтобы запросить записи, необходимо задать: имя результирующего набора; номер записи в результирующем наборе, с которой начинается извлечение; количество извлеченных записей; формат представления данных и максимальную длину записи, доступную для извлечения.

Рассмотрим модель извлечения и представления данных, принятую в Z39.50. Идеология максимального абстрагирования от структур реальных баз данных приводит к весьма изощренной схеме извлечения данных, описанной в стандарте Z39.50. Каждой базе данных приписывается одна или несколько схем.

Схема данных и абстрактная структура записи

Схема базы данных представляет собой взаимное соглашение между клиентом и сервером об информации, содержащейся в записях базы данных**,** которое позволяет в дальнейшем отбирать часть этой информации в соответствии со спецификацией элемента.

Схема определяет *абстрактную структуру записи***.** Это первичный элемент схемы базы данных, представляющий собой дерево элементов, специфицируемых тэгами (*tag*) из стандартных наборов тэгов (*tagSet*). При применении абстрактной структуры записи к записи базы данных получается абстрактная запись базы данных.

Абстрактная запись базы данных — абстрактное представление информации, содержащейся в записи базы данных. Для формирования абстрактной записи нужно применить определенную в схеме абстрактную структуру записи к записи базы данных.

Спецификация элемента преобразует абстрактную запись базы данных в другой экземпляр абстрактной записи (преобразование может быть нулевым). По спецификации элемента из абстрактной записи отбираются эле-

менты, а также, возможно, указываются формы варианта для этих элементов.

Сервер применяет к этой абстрактной записи *синтаксис записи* для представления извлеченных записей, в результате чего получается структура, подлежащая пересылке (извлеченная запись).

Теперь на конкретном примере рассмотрим вышеопределенные понятия схемы базы данных и абстрактной структуры записи. Предположим, существуют два музея (музейных информационных систем), в которых независимо ведется учет музейных предметов. Учет ведется в различных СУБД, информация классифицируется по следующим полям.

Таблица 11

Список полей баз данных двух музейных систем

Museum1	**Museum2**
Identifier	ObjectID
Title	ObjectTitle
Type	ObjectType
Creator	CreatorName
	CreatorDateOfBirth
	CreatorDateOfDeath
	CreatorRole
Date	DateOfOrigin
Place	PlaceOfOrigin
Material	
Technique	
Dimensions	
Period	StylePeriod
Keywords	Subject
Description	

Очевидно, что происходит накопление однотипной информации, но структуры данных в системах отличаются. Построим некую обобщающую модель данных, в которой зафиксируется суть накапливаемой информации. Эта модель должна включать позиции, представленные в табл. 12.

Таблица 12

Обобщающая модель данных

Идентификационный номер предмета
Название предмета
Типология
Информация о создателе (имя, дата рождения, дата смерти, роль)
Дата создания
Место создания
Материал
Техника
Размеры
Культурный период
Ключевые слова
Краткое описание предмета

В иерархической схеме это будет выглядеть следующим образом:

```
Идентификатор
Название
Типология
Создатель
        Имя
        Дата рождения
        Дата смерти
        Роль
Дата создания
Место создания
Материал
Техника
Размеры
Культурный период
Ключевые слова
Краткое описание
```

К этой схеме нужно добавить дополнительные характеристики, такие как обязательность, повторяемость полей и поисковые атрибуты. Пример приведен в табл. 13.

Таблица 13

Схема данных

Название поля	Обязательность	Повторяемость	Поисковые атрибуты
Идентификатор	обязательно	не повторяется	2024
Название	обязательно	не повторяется	4,2033,2047
Типология	обязательно	не повторяется	1031,2032
Создатель	обязательно	не повторяется	1003,2041
Имя	обязательно	не повторяется	2035,2046
Дата рождения	не обязательно	не повторяется	2036
Дата смерти	не обязательно	не повторяется	2037
Роль	не обязательно	не повторяется	2014
Дата создания	обязательно	не повторяется	58,2022,2048
Место создания	обязательно	не повторяется	2023,2049
Материал	не обязательно	повторяется	2008
Техника	не обязательно	повторяется	2012
Размеры	не обязательно	не повторяется	2074
Культурный период	обязательно	повторяется	2017
Ключевые слова	обязательно	повторяется	2040
Краткое описание	не обязательно	не повторяется	62

В последнем столбце проставлены соответствующие значения поисковых Use атрибутов по CIMI-1. Таким образом, мы определили модель данных, которая называется в Z39.50 *схемой данных*.

Следующий шаг — определение *абстрактной структуры записи*. Для этого воспользуемся наборами тэгов Generic Tag Set (2), набор элементов которого приведен ниже в таблице 14, и CIMI Tag Set (5), набор элементов которого будет приведен позже при рассмотрении профиля CIMI в таблице 19.

Таблица 14

Набор тэгов Generic Tag Set (TagSet-G)

тэг	элемент	тэг	элемент	тэг	элемент	тэг	элемент
1	title	11	postalAddress	21	subject	31	publisher
2	author	12	networkAddress	22	resourceType	32	contributor
3	publication-Place	13	eMailAddress	23	city	33	source
4	publicationDate	14	phoneNumber	24	stateOrProvince	34	coverage
5	documentId	15	faxNumber	25	zipOrPostalCode	35	private
6	abstract	16	country	26	cost	36	data-baseName
7	name	17	description	27	format	37	recordId
8	dateTime	18	time	28	identifier		
9	displayObject	19	DocumentContent	29	rights		
10	organization	20	language	30	relation		

В таблице 15 приведены элементы и тэги, используемые в абстрактной структуре записи.

Таблица 15

Используемые элементы и тэги

(2,7) - Name	(5,5) - MaterialMedium	(5,11) — PlaceOfOrigin	(5,31) - ObjectName
(2,17) - Description	(5,8) - DateOfBirth	(5,12) - ProcessTechnique	(5,32) - ObjectTitle
(5,2) - Subject	(5,9) - DateOfDeath	(5,13) — Dimensions	(5,36) - CreatorInfo
(5,3) - ObjectID	(5,10) - Role	(5,14) — StylePeriod	(5,45) - DateOfOrigin

Определим меточную абстрактную структуру записи.

Таблица 16

Абстрактная структура записи

Тэговый путь	Элемент
(5,3)	ObjectID
(5,32)	ObjectTitle
(5,31)	ObjectType
(5,36)	Creator

(5,36)/(2,7)	CreatorName
(5,36)/(5,8)	CreatorDateOfBirth
(5,36)/(5,9)	CreatorDateOfDeath
(5,36)/(5,10)	CreatorRole
(5,45)	DateOfOrigin
(5,11)	PlaceOfOrigin
(5,14)	Period
(5,5)	Material
(5,12)	Technique
(5,13)	Dimensions
(5,2)	Keywords
(2,17)	Description

Первая цифра в скобках означает номер набора тэгов, а вторая — номер тэга в наборе. Каждая строчка в определении абстрактной структуры записи имеет тэговый путь (TagPath), именована и называется элементом. Элементы могут быть простыми (ObjectTitle) и сложными (Creator).

Теперь вернемся к двум музейным системам, которым требуется на уровне извлеченных записей объединить свои базы данных. Для этого им необходимо просто выдавать записи в соответствии с определенной схемой. В каждом музее потребуется установление соответствия элементов абстрактной структуры полям реальных баз данных и соответственное их наполнение.

Формат GRS-1

Рассмотренные схема данных и абстрактная структура записи определяют лишь логическую структуру записи. Физическая реализация этой структуры остается неопределенной. Стандарт Z39.50 требует, чтобы записи из внутреннего представления отображались во внешние структуры, которые и пересылаются по сети. Для внешних представлений, которые стандартизуются в классе recordSyntax {Z39.50 5}, физическая реализация становится определяющей.

Существует всего один универсальный формат внешнего представления (recordSyntax), который однозначно отображает абстрактную структуру записи в экспортируемую внешнюю физическую структуру. Он называется GRS-1 (Generic Record Syntax) OID {Z39.50 5 105}.

Полное ASN.1 определение записи в формате GRS-1 можно найти в приложении к документации по Z39.50 [5,6]. Если в этом определении оста-

вить основное, исключив метаинформацию (metaData) и варианты (applied-Variant), то останется:

```
GenericRecord ::= SEQUENCE OF TaggedElement
TaggedElement ::= SEQUENCE {
   tagType          [1] IMPLICIT INTEGER OPTIONAL,
   tagValue         [2] StringOrNumeric,
   tagOccurrence    [3] IMPLICIT INTEGER OPTIONAL,
   content          [4] ElementData,
   metaData         [5] IMPLICIT ElementMetaData OP-
TIONAL,
   appliedVariant   [6] IMPLICIT Variant OPTIONAL}

ElementData ::= CHOICE{
   octets                  OCTET STRING,
   numeric                 INTEGER,
   date                    GeneralizedTime,
   ext                     EXTERNAL,
   string                  InternationalString,
   trueOrFalse             BOOLEAN,
   oid                     OBJECT IDENTIFIER,
   intUnit             [1] IMPLICIT IntUnit,
   elementNotThere     [2] IMPLICIT NULL, -- element
requested but not there
   elementEmpty        [3] IMPLICIT NULL, -- element
there, but empty
   noDataRequested     [4] IMPLICIT NULL, -- variant
request said 'no data'
   diagnostic          [5] IMPLICIT EXTERNAL,
   subtree             [6] SEQUENCE OF TaggedElement
}
```

Итак, упрощенно можно полагать, что запись в формате GRS-1 — это последовательность элементов (taggedElement), каждый их которых содержит тип тэга (tagType, т. е. номер tagSet), значение тэга (tagValue), вхождение тэга (tagOccurrence) и данные (content). Значение тэга может быть числовым или текстовым. Данные — это или значение, или вложенный элемент (taggedElement) с описанной структурой (определение subtree ничем не отличается от определения GenericRecord). Таким образом, GRS-1 представляет собой последовательность элементов со структурой, определенной в терминах, аналогичных определению абстрактной структуры записи и схемы данных, с бесконечным количеством вложений.

5. ПРОФИЛЬ CIMI

Теперь рассмотрим более подробно, как происходит доступ к информации о культурном наследии по протоколу Z39.50 в соответствии с профилем CIMI (OID 1.2.840.10003.3.8.) [4,10].

Профиль CIMI определяет подмножество характеристик Z39.50, опций и параметров, необходимых для поддержки функциональности и требований пользователя при поиске и извлечении информации о культурном наследии. Профиль CIMI оперирует со следующими объектами Z39.50.

Таблица 17

Объекты Z39.50 профиля CIMI

Наборы поисковых атрибутов:	Bib-1	1.2.840.10003.3.1
	CIMI-1	1.2.840.10003.3.8
Диагностика:	Bib-1	1.2.840.10003.4.1
Форматы:	GRS-1	1.2.840.10003.5.105
	SUTRS	1.2.840.10003.5.101
	Usmarc	1.2.840.10003.5.10
Схемы:	Digital Collections Schema	1.2.840.10003.13.3
	CIMI Schema	1.2.840.10003.13.5
Наборы тэгов:	TagSet-M (Metadata TagSet)	1.2.840.10003.14.1
	TagSet-G (Generic TagSet)	1.2.840.10003.14.2
	Collections TagSet	1.2.840.10003.14.5
	CIMI TagSet	1.2.840.10003.14.6

На основе этих объектов определены модели поиска и извлечения данных с музейной информацией. Полное описание профиля CIMI достаточно объемно и здесь приводиться не будет [10]. Ниже будут отмечены только некоторые аспекты, характерные для этого профиля.

Поиск

Для проведения пользователем поиска во многих базах данных на нескольких серверах необходимо стандартизировать поисковое выражение таким образом, чтобы клиент и сервер могли обмениваться информацией недвусмысленным образом. Эта задача решается путем определения набора атрибутов, который определяет множество пунктов доступа и дополнитель-

ную информацию, используемую для характеристики поисковых термов, а также путем представления поисковой строки стандартным образом.

Набор атрибутов CIMI-1

Профиль CIMI определяет стандартный *набор атрибутов CIMI-1* для поиска информации о культурном наследии, поддержка которого является обязательной для CIMI клиентов и серверов. Набор CIMI-1 использует шесть типов атрибутов, определенных в наборе Bib-1 (типы атрибутов Use, Relation, Position, Structure, Truncation и Completeness) [7]. CIMI-1 также определяет два новых типа атрибутов: тип 101 — Authority (авторитетный источник) и тип 102 — Charset (набор символов).

Тип атрибутов *Authority* содержит значения, идентифицирующие авторитетный источник, из которого взят термин. В качестве примера значений Authority атрибутов можно перечислить следующие: Non-authoritative, Local-to-server, USMARC, LCSH (Library of Congress Subject Headings), AAT (Art & Architecture Thesaurus), Dictionarium Museologicum, ISO Documentation и так далее.

Тип атрибутов *Charset* включает значения, идентифицирующие набор символов, используемый для кодировки термина. В качестве примера значений Charset атрибутов можно привести следующие: 7-bit US-ASCII, ISO 8859-1 (Latin-1), ISO 8859-5 (Cyrillic), ISO 8859-6 (Arabic) и так далее.

Что касается Use атрибутов, определенных в CIMI-1, то они частично получены из пунктов доступа CIMI (CIMI Access Points) [8], а также из других руководств и стандартов по информации о культурном наследии. Использование Use атрибутов CIMI-1 позволяет клиенту расширить поиск музейной информации по специфическому содержанию (например, по названию и типологии объекта, сведениям о создателе, дате и месте происхождения объекта, культурному периоду, материалу и технике, размерам, данным о хранении и т.д.) стандартным путем, который может быть понят сервером. CIMI-1 также содержит небольшое подмножество (14 значений с номерами в промежутке от 4 до 1032) Use атрибутов набора Bib-1 (title, ISBN, ISSN, date of publication, abstract, author и т.д.). Эти атрибуты включены для обеспечения CIMI клиентам базисного поиска (автор–название–ключевые слова) в библиографических базах данных, а также для поддержки поиска на CIMI серверах библиографическими клиентами Z39.50, не поддерживающими профиль CIMI.

Набор атрибутов CIMI-1 был разработан консорциумом CIMI при изучении существующих стандартов и систем после проведения анализа запросов пользователей к музейным коллекциям. Он отражает в себе соглашения

с широким музейным сообществом о наборе пунктов доступа, которые должна поддерживать система. Таким образом, набор CIMI-1 предоставляет механизм совместного понимания при поиске для клиента и сервера. Например, когда пользователь отправляет запрос на извлечение информации о «происхождении произведения», база данных сервера может иметь или не иметь «происхождение произведения» как отдельный пункт доступа. В этом случае необходимо, чтобы такой запрос был корректно отображен сервером на соответствующие поля или индексы локальной базы данных. Сервер, поддерживающий этот профиль, может понять, когда он получает запрос на «происхождение произведения», поскольку запрос представлен и выражен в стандартной форме набора атрибутов CIMI-1.

Для переадресации запросов профиль использует набор элементов метаданных Дублинского ядра (Dublin Core Metadata) [11]. Use атрибуты, ассоциированные с элементами Дублинского ядра (DC-title, DC-creator, DC-subject, DC-description и т.д.), могут выразить запрос на поиск в терминах пунктов доступа, представленных или охарактеризованных элементами Дублинского ядра.

Локальные реализации могут определять Use атрибуты для локального использования и присваивать им значения, находящиеся в промежутке 5000-7999.

Полная спецификация набора атрибутов CIMI-1 приведена в приложении к документации по CIMI профилю (APPENDIX A: CIMI-1 Attribute Set). Ниже приведен список Use атрибутов CIMI-1, информация о семантике которых доступна в приложении к документации (APPENDIX C: Semantics for Use Attributes and Schema Elements) [10].

Таблица 18

Use атрибуты набора CIMI-1

Значение	Название атрибута	Значение	Название атрибута
4	title	2041	creatorGeneral
7	ISBN	2042	associationGeneral
8	ISSN	2043	objectLanguage
31	date of publication	2044	condition
32	date of acquisition	2045	physicalDescription
54	code language	2046	who
58	name geographic	2047	what
62	abstract	2048	when
1003	author	2049	where
1004	personal author	2051	DC-title
1016	any	2052	DC-creator
1018	publisher	2053	DC-subject
1031	material type	2054	DC-description
1032	doc-id	2055	DC-publisher
2000	award	2056	DC-contributors
2002	collection	2057	DC-date
2004	copyrightRestriction	2058	DC-type
2005	creditLine	2059	DC-format
2007	inscriptionMark	2060	DC-identifier
2008	materialMedium	2061	DC-source
2009	creatorNationalityCulturеRace	2062	DC-language
2012	processTechnique	2063	DC-relation
2014	creatorRole	2064	DC-coverage
2017	stylePeriod	2065	DC-rights
2020	image	2070	fieldCollector
2022	dateOfOrigin	2071	dateCollected
2023	placeOfOrigin	2072	agePeriod
2024	objectID	2073	typeSpecimen
2026	owner	2074	dimensions
2027	repositoryName	2075	quantity
2028	repositoryPlace	2076	relatedObjects
2029	provenance	2077	resource
2030	contentGeneral	2078	wallTextLabel
2032	objectName	2079	administrativeEventGeneral
2033	objectTitle	2080	administrator
2034	relatedTextualReferences	3000	protectionStatus
2035	creatorName	3001	protectionDate
2036	creatorDateOfBirth	3003	spatialReferencingSystem
2037	creatorDateOfDeath	3004	x-coordinateInReferencingSystem
2038	contextHistorical	3005	y-coordinateInReferencingSystem
2039	contextArchaelogical	3007	Address
2040	subjectContent	3009	PeriodName

CIMI сервера должны предусматривать ситуацию, когда поисковый запрос содержит только один поисковый терм, а атрибуты не присутствуют в запросе. В этом случае в CIMI профиле рекомендуется использовать следующие значения по умолчанию для атрибутов каждого типа. Если запрос содержит только поисковый терм без указания Use атрибута, для Use атрибута принимается значение по умолчанию, равное «Any» (1016). Что касается атрибутов типов 2-6, следующие значения рекомендуются по умолчанию: для типа Relation — «equal» (3), типа Position — «any position in field» (3), типа Structure — «word» (2), типа Truncation — «do not truncate» (100) и типа Completeness — «complete field» (3). Тип Authority не имеет значения по умолчанию, сервер интерпретирует его отсутствие в запросе клиента просто как отсутствие данных по этому поводу и обрабатывает запрос по своему усмотрению. Поддержка в запросе логических операторов AND и OR для CIMI клиентов и серверов является обязательной.

Выбор

Для проведения поиска информации во многих базах данных на нескольких серверах необходимо обеспечить выполнение следующих двух требований. Во-первых, клиенты и серверы должны быть способны обмениваться записями из баз данных (или элементами записей) в форматах, которые они могут распознавать и обрабатывать. Во-вторых, клиенты и серверы должны иметь одинаковое понимание элементов в этих базах данных и возможность идентифицировать эти элементы недвусмысленно для выбора информации, которую необходимо получить. Спецификации выбора позволяют клиенту и серверу обмениваться информацией о записях в базе данных для извлечения полной записи или определенных единиц информации (т.е. одной или более групп полей базы данных).

Профиль CIMI реализует эту возможность путем определения набора тэгов CIMI (CIMI Tag Set), схемы CIMI (CIMI Schema) и абстрактной структуры записи (Abstract Record Structure — ARS) для извлечения записей. Аналогично, как и для набора атрибутов CIMI-1, обсужденных выше, схема CIMI и соответствующая абстрактная структура записи служат «языком взаимопонимания» для обмена между клиентом и сервером при извлечении информации.

Схема CIMI и набор тэгов CIMI Tag Set

Схема CIMI абстрактно определяет единицы информации, которые могут присутствовать в записях локальных баз данных объектов, изображений

с присоединенным текстом и каталожных записях. Схема не указывает, как поле названо в реальной базе данных, а предлагает стандартный путь для именования этих элементов или полей. Всякая локальная база данных применяет свои термины для обозначений полей базы данных и их структуры. Схема обеспечивает абстрактное представление этих баз данных: в этом абстрактном виде поля базы данных пронумерованы как элементы схемы, каждый элемент которой имеет уникальное имя, уникальную цифровую метку (тэг) и определение. Схема также показывает структурную организацию этих элементов в абстрактную структуру записи.

Например, схема CIMI определяет элемент *dateOfOrigin*. Локальная база данных может иметь одно или более полей, относящихся к «дате создания объекта». Поскольку семантика предлагается для каждого элемента схемы CIMI, администратор базы данных знает, что в случае, если клиентом запрошен элемент *dateOfOrigin*, то должна быть возвращена информация, относящаяся к «дате создания объекта».

CIMI схема использует элементы, определенные в наборе тэгов CIMI Tag Set, так же как и элементы из других зарегистрированных наборов тэгов.

Ниже приведен *набор тэгов CIMI Tag Set* (OID=1.2.840.10003.14.6), содержащий значения тэгов и названия элементов. Семантика этих элементов приведена в приложении к документации по профилю CIMI (Appendix C -- Semantics for Use Attributes and Schema Elements).

Таблица 19

Набор тэгов CIMI Tag Set

Тэг	Элемент	Тэг	Элемент	Тэг	Элемент
1	RepositoryName	25	content	50	associationGeneral
2	Subject	26	repositoryPlace	51	objectLanguage
3	ObjectID	28	mrObject	52	condition
4	nationalityCultur-eRace	29	rendition	53	physicalDescription
5	MaterialMedium	30	resource	54	wallTextLabel
7	CreditLine	31	objectName	55	protectionStatus
8	DateOfBirth	32	objectTitle	56	protectionDate
9	DateOfDeath	33	bibliographicTitle	57	spatialReferencing-System
10	Role	35	relatedTextual-References	58	xCoordinateInSpatial-ReferencingSystem
11	PlaceOfOrigin	36	creatorInfo	59	yCoordinateInSpatial-ReferencingSystem
12	ProcessTechnique	38	owner	60	fieldCollector
13	Dimensions	39	contentGeneral	61	dateCollected

14	StylePeriod	41	place	62	agePeriod
15	Provenance	42	event	63	typeSpecimen
16	RelatedObjects	43	activity	64	address
17	Quantity	45	dateOfOrigin	65	periodName
18	Award	46	contextHistorical	66	administrativeEvent
20	Collection	47	contextArchaeologi-cal	67	administrativeEvent-Type
22	InscriptionMark	48	copyrightRestric-tion	68	administrativeEvent-General
24	Association	49	creatorGeneral	69	administrator

CIMI схема определяет также пять структурных типов данных (элементов): *CreatorInfo, MrObject, Rendition, MoreInfo, AdministrativeEvent*. Схема позволяет извлекать одно или более «изображений», ассоциированных с записью об объекте (где под «изображением» понимается любой тип электронного ресурса, включая видео, аудио, графику). Каждому экземпляру «изображения» соответствует элемент схемы *mrObject*. Поскольку один и тот же электронный ресурс может быть представлен в нескольких вариантах (например, изображения различных размеров), схема содержит понятие о «представлении» (*Rendition),* которое является специфической версией «изображения». Каждому имеющемуся варианту «изображения» соответствует элемент *Rendition,* подэлемент элемента *mrObject*. Элемент *Rendition* включает подэлемент *Resource,* который содержит или URL ресурса, или непосредственно сам электронный ресурс. Таким образом, сервер может вернуть клиенту одно или более «изображений», так же как и одно или более видов (представлений) каждого «изображения». Более того, для каждого изображения и наглядного представления может быть извлечена специфическая описательная информация.

Абстрактная структура записи CIMI

CIMI схема описывает *абстрактную структуру записи*, определяющую размещение и порядок элементов в извлеченной записи. Поскольку CIMI профиль является дополнительным к профилю для доступа к цифровым коллекциям (Digital Collections Profile) [9], схема данных CIMI и абстрактная структура записи основаны на использовании схемы Digital Collections. Абстрактная структура записи CIMI поддерживает структуру записи Digital Collections и вложена в элемент actualDO записи Digital Collections (object descriptive record).

Термин абстрактная структура записи обычно используется в контексте специфической схемы. В CIMI профиле абстрактная структура записи вы-

ходит за пределы CIMI схемы. Абстрактная структура записи специфицирует использование элементов из наборов тэгов tagSet-M, tagSet-G, Collections Tag Set и CIMI Tag Set. Следующие типы тэгов (1-5) используются в ней для извлеченной записи.

Таблица 20

Типы тэгов в абстрактной структуре записи CIMI

Тип тэгов	Определение
1	Элементы из tagSet-M. Сервер может включать их по своему выбору, а клиент может их игнорировать (кроме элемента schemaIdentifier).
2	Элементы из tagSet-G. Сервер может включать элементы, не перечисленные в абстрактной структуре записи, а клиент может их игнорировать.
3	Зарезервирован для тэгов, локально определенных сервером. Серверы, посылающие строковые тэги для локально определенных элементов, должны использовать 3 тип тэгов для их идентификации. Строковые тэги следует использовать только тогда, когда имеющиеся в наличии элементы из tagSet-M, tagSet-G, Collections TagSet и CIMI TagSet не адекватны.
4	Элементы из Collections Tag Set.
5	Элементы из CIMI Tag Set.

Абстрактная структура извлеченной записи, определенная в CIMI профиле, поддерживает три уровня семантической интероперабельности путем разделения записи на три уровня элементов: общий уровень (Generic Level), уровень схемы цифровых коллекций (Digital Collections Schema Level) и уровень CIMI схемы (CIMI Schema Level). Это разделение позволяет обеспечить семантическую интероперабельность между CIMI серверами и Z39.50 клиентами, не поддерживающими CIMI профиль.

Чтобы обеспечить эти три уровня семантической интероперабельности, необходимо добавить элемент *schemaIdentifier* (из tagSet-M) в извлеченную запись. На общем уровне семантической интероперабельности абстрактная структура записи предполагает использование тэгов типов 1, 2 и 3. На уровне Digital Collections — использование тэгов типов 1, 2 ,3 и 4, а на уровне CIMI — тэгов типов 1, 2 ,3, 4 и 5.

В нижеприведенных таблицах используются следующие обозначения для вхождения элементов: [0,1] (не обязательно, не повторяется), 0+ (не обязательно, повторяется), 1 (обязательно, не повторяется), 1+ (обязательно, повторяется).

1. Общий уровень (Generic Level) семантической интероперабельности: в начале извлеченной записи могут присутствовать элементы из tagSet–M и tagSet–G. Серверы и клиенты Z39.50 могут распознавать и обрабатывать эти элементы, даже если они не поддерживают ни одной специализированной схемы. Таким образом, включение общих элементов на верхнем уровне извлеченной записи позволяет любым клиентам Z39.50 производить поиск в базах данных, содержащих описательные записи коллекций и объектов, и частично их обрабатывать.

Таблица 21

Общий уровень абстрактной структуры записи

Тэговый путь	Элемент	Вхождение
(1,14)	localControlNumber	1
(2,1)	title	0+
(2,2)	creator	0+
(2,32)	contributor	0+
(2,8)	date	0+
(2,170)	description	0+
(2,28)	identifier	0+
(2,22)	type	0+
(2,20)	language	0+
(2,21)	subject	0+
(2,31)	publisher	0+
(2,27)	format	0+
(2,33)	source	0+
(2,30)	relation	0+
(2,34)	coverage	0+
(2,29)	rights	0+

2. Уровень схемы цифровых коллекций (Digital Collections Schema Level) семантической интероперабельности: принимается схема Digital Collections (указывается с помощью schemaIdentifier), и поддерживающие ее клиенты Z39.50 могут распознавать и обрабатывать эти элементы. Таким образом, клиенты, поддерживающие схему Digital Collections, могут выполнять поиск в базах данных, предоставляемых CIMI серверами. Они могут извлекать эти записи, но не могут обрабатывать их полностью.

Таблица 22

Тэговый путь	Элемент	Вхождение
(1,1)	schemaIdentifier	1
(4,1)	typeOfDescriptiveRecord	1
(4,4)	objectInfo	1
(4,4)(4,12)	typeOfObject	1
(4,4)(4,13)	categoryOfObject	[0,1]
(4,4)(4,14)	digitalObject	1
(4,4)(4,14)(4,29)	actualDO	1

*Табл. 22. У*ровень Digital Collections абстрактной структуры записи

3. Уровень схемы CIMI (CIMI Schema Level) семантической интероперабельности: принимается схема CIMI (указывается с помощью *schemaIdentifier*), и Z39.50 клиенты, ее поддерживающие, могут распознавать и обрабатывать остальные элементы записи (т.е. полностью обрабатывать целую извлеченную запись).

Таблица 23

Уровень CIMI абстрактной структуры записи

Тэговый путь	Элемент	Вхождение
(4,4)(4,14)(4,29)(1,1)	SchemaIdentifier	1
(4,4)(4,14)(4,29)(5,31)	ObjectName	0+
(4,4)(4,14)(4,29)(5,32)	ObjectTitle	1+
(4,4)(4,14)(4,29)(5,33)	bibliographicTitle	1+
(4,4)(4,14)(4,29)(5,49)	creatorGeneral	0+
(4,4)(4,14)(4,29)(5,36)	creatorInfo	0+
(4,4)(4,14)(4,29)(5,60)	fieldCollector	0+
(4,4)(4,14)(4,29)(5,1)	repositoryName	0+
(4,4)(4,14)(4,29)(5,26)	repositoryPlace	0+
(4,4)(4,14)(4,29)(5,38)	owner	0+
(4,4)(4,14)(4,29)(5,7)	creditLine	0+
(4,4)(4,14)(4,29)(5,2)	subject	0+
(4,4)(4,14)(4,29)(5,3)	objectID	0+
(4,4)(4,14)(4,29)(5,5)	materialMedium	0+
(4,4)(4,14)(4,29)(5,12)	processTechnique	0+
(4,4)(4,14)(4,29)(5,13)	dimensions	0+
(4,4)(4,14)(4,29)(5,11)	placeOfOrigin	0+
(4,4)(4,14)(4,29)(5,45)	dateOfOrigin	0+

(4,4)(4,14)(4,29)(5,61)	dateCollected	0+
(4,4)(4,14)(4,29)(5,62)	agePeriod	0+
(4,4)(4,14)(4,29)(5,63)	typeSpecimen	0+
(4,4)(4,14)(4,29)(5,14)	stylePeriod	0+
(4,4)(4,14)(4,29)(5,65)	periodName	0+
(4,4)(4,14)(4,29)(5,15)	provenance	0+
(4,4)(4,14)(4,29)(5,17)	quantity	0+
(4,4)(4,14)(4,29)(5,18)	award	0+
(4,4)(4,14)(4,29)(5,20)	collection	0+
(4,4)(4,14)(4,29)(5,22)	inscriptionMark	0+
(4,4)(4,14)(4,29)(5,51)	objectLanguage	0+
(4,4)(4,14)(4,29)(5,52)	condition	0+
(4,4)(4,14)(4,29)(5,53)	physicalDescription	0+
(4,4)(4,14)(4,29)(5,55)	protectionStatus	0+
(4,4)(4,14)(4,29)(5,56)	protectionDate	0+
(4,4)(4,14)(4,29)(5,57)	spatialReferencingSystem	0+
(4,4)(4,14)(4,29)(5,58)	xCoordinateInSpatialReferencingSystem	0+
(4,4)(4,14)(4,29)(5,59)	yCoordinateInSpatialReferencingSystem	0+
(4,4)(4,14)(4,29)(5,64)	address	0+
(4,4)(4,14)(4,29)(5,16)	relatedObjects	0+
(4,4)(4,14)(4,29)(5,35)	relatedTextualReferences	0+
(4,4)(4,14)(4,29)(5,50)	associationGeneral	0+
(4,4)(4,14)(4,29)(5,24)	association	0+
(4,4)(4,14)(4,29)(5,39)	contentGeneral	0+
(4,4)(4,14)(4,29)(5,25)	content	0+
(4,4)(4,14)(4,29)(5,46)	contextHistorical	0+
(4,4)(4,14)(4,29)(5,47)	contextArchaelogical	0+
(4,4)(4,14)(4,29)(5,48)	copyrightRestriction	0+
(4,4)(4,14)(4,29)(5,54)	wallTextLabel	0+
(4,4)(4,14)(4,29)(2,9)	displayObject	0+
(4,4)(4,14)(4,29)(5,68)	administrativeEventGeneral	0+
(4,4)(4,14)(4,29)(5,66)	administrativeEvent	0+
(4,4)(4,14)(4,29)(5,69)	administrator	0+
(4,4)(4,14)(4,29)(5,28)	mrObject	0+

Наборы элементов b , mb , f

Сервер создает *результирующий набор* в ответ на запрос. Результирующий набор — это множество указателей на записи в одной или более базах

данных. *Именованные наборы элементов* идентифицируют группы элементов в записи базы данных, возвращаемой клиенту сервером. CIMI профиль определяет следующие три именованные набора элементов для абстрактной структуры извлеченной записи: ***b , mb , f .***

Набор элементов b (brief record**)** предназначен для извлечения краткой формы записи базы данных, составленной из элементов tagSet–G верхнего уровня абстрактной структуры извлеченной записи, который соответствует основному (общему) уровню поиска и извлечения при использовании элементов Dublin Core Metadata. Что касается элементов, составляющих набор элементов **b,** они в точности совпадают с набором элементов общего уровня абстрактной структуры записи и уже были приведены выше в табл. 21. Сервер может также включать в запись другие элементы из tagSet–G и tagSet–M.

Набор элементов mb (museum **b**rief record**)** предназначен для извлечения краткой формы записи базы данных согласно принятым в музейных системах стандартам. Цель этой краткой формы — предоставить CIMI клиентам достаточное количество элементов для построения краткой записи *(tombstone record)*. Набор элементов **mb** включает элементы уровней Digital Collections и CIMI абстрактной структуры записи, которые приведены в таблицах 22 и 23. Сервер может также включать в запись элементы из tagSet–G и tagSet–M.

Набор элементов f (full record**)** включает все имеющиеся элементы записи базы данных. Элементы записи базы данных, которые могут быть помечены имеющимися тэгами из tagSet–G, tagSet–M, tagSet–Collections и tagSet–CIMI, должны быть помечены соответствующим образом. Дополнительные элементы, возвращаемые сервером, должны быть или включены в элемент *displayObject,* или должны использоваться строковые тэги из третьего типа тэгов для локально определенных элементов. Строковые тэги и содержащиеся в *displayObject* данные клиент не обрабатывает, а предоставляет пользователю в неизмененном виде.

Передача

Схема CIMI и ассоциированная с ней абстрактная структура записи описывают, как должны быть однозначно помечены сервером элементы/поля базы данных. Передача элементов от сервера к клиенту требует еще одного набора спецификаций. Z39.50 использует понятие синтаксиса записей для определения того, каким образом сервер упаковывает элементы базы данных для отсылки клиенту. Синтаксис записи предписывает серверу, каким

образом должны быть отформатированы элементы/поля базы данных перед отсылкой их клиенту.

Общий синтаксис записи GRS-1 (Generic Record Syntax) Z39.50 позволяет серверу использовать произвольно структурированные данные. Профиль CIMI требует поддержки только синтаксиса записи GRS-1, обязательной является поддержка следующих параметров GRS-1: tagType, tagValue, tagOccurrence, content, appliedVariant.

Для обеспечения интероперабельности между библиотеками и музеями сервера могут поддерживать другие синтаксисы записи, включая USMARC и SUTRS (Simple Unstructured Text Record Syntax).

Схема CIMI может быть использована и вне протокола Z39.50. Тогда, когда работа происходит за рамками профиля CIMI, можно создавать и передавать записи баз данных, соответствующих схеме CIMI, в других форматах, таких как, например, XML.

6. ПРИМЕР РАСПРЕДЕЛЕННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ НА БАЗЕ Z39.50

В заключение приведем пример распределенной информационной системы на базе Z39.50, схема которой приведена на рис. 2.

Центральное место в информационной системе занимают несколько серверов Z39.50, каждый из которых обеспечивает доступ к своим базам данных. Все серверы Z39.50 связаны механизмом перенаправления запросов.

В качестве клиентского программного обеспечения для простого доступа к информационной системе можно использовать шлюз Z39.50-WWW, представляющий собой CGI-приложение с функциями клиента Z39.50. Для этого система должна содержать WWW-сервер, на котором исполняется программа шлюза WWW-Z39.50.

Доступ пользователей к информационной системе может осуществляться из обычных WEB-броузеров (клиент WWW) через шлюз WWW-Z39.50 или из специализированных рабочих мест по протоколу Z39.50 (клиент Z39.50).

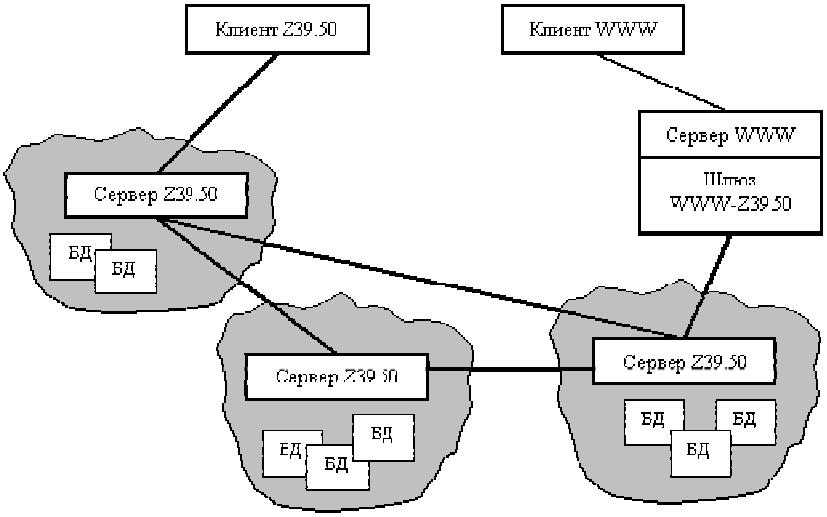


Рис. 2. Схема распределенной информационной системы на базе Z39.50

ЗАКЛЮЧЕНИЕ

Профиль CIMI отражает набор спецификаций при использовании Z39.50 для поиска и извлечения информации о культурном наследии. Он также предоставляет две значительных области стандартизации, которые могут быть полезны вне области применения Z39.50.

Во-первых, набор атрибутов CIMI-1 определяет большой набор пунктов доступа, который может быть использован для поиска информации о культурном наследии. Поскольку этот набор пунктов доступа был получен в результате эмпирических исследований и обсуждений с членами музейного сообщества, он может рассматриваться как представление общего набора пунктов доступа, полезных в области информации о культурном наследии.

Во-вторых, схема CIMI предоставляет стандартный список элементов баз данных и организации этих элементов для обмена информацией о культурном наследии. Стандартный список может быть использован как конвертор или метаязык, для того чтобы пометить элементы локальной базы данных и обмена этими элементами с другими системами.

Z39.50 как протокол обмена <компьютер-компьютер> использует эти структуры, для того чтобы сделать возможным интероперабельный поиск и

извлечение информации. В контексте приложения к информации о культурном наследии профиль CIMI определяет использование взаимопонятных атрибутов и элементов схемы для стабильного извлечения информации посредством Z39.50.

Возможно, в скором времени и музейное сообщество России подключится к программе предоставления доступа к своим информационным ресурсам по протоколу Z39.50 с использованием профиля CIMI.

СПИСОК ЛИТЕРАТУРЫ

1. Жижимов О.Л. Введение в Z39.50. — Новосибирск: Изд-во НГОНБ, 2000.
2. Жижимов О.Л., Мазов Н.А. О доступе к информационным ресурсам по культурному наследию по протоколу Z39.50 // Матер. конф. EVA'2000. «Электронная конвергенция: новые технологии в музеях, галереях, библиотеках и архивах», 30 окт.–3 нояб. 2000 г. — М.: Центр ПИК Минкультуры РФ, 2000. — 08-2-1–08-2-2.
3. Касьянов В.Н., Несговорова Г.П., Волянская Т.А. Виртуальный музей истории информатики в Сибири. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 169–181.
4. Мазов Н.А., Жижимов О.Л. Профиль Z39.50-CIMI как основа интеграции информационных ресурсов по культурному наследию // Матер. конф. EVA'2003. «Информация для всех: культура и технологии информационного общества», 1–5 дек. 2003 г. — М.: Центр ПИК Минкультуры РФ, 2003.
5. ANSI/NISO Z39.50-1995. Information Retrieval (Z39.50): Application Service Definition and Protocol Specification. // Z39.50 Maintenance Agency Official Text for Z39.50-1995. — 156 p. <http://lcweb.loc.gov/z3950/agency/1995doce.html>
6. ANSI/NISO Z39.50-2003. Information Retrieval (Z39.50): Application Service Definition and Protocol Specification. // NISO Press, Bethesda, Maryland, U.S.A. — 276 p. <http://www.niso.org/standards/resources/Z39-50-2003.pdf>
7. Attribute Set Bib-1 (Z39.50-1995): Semantics. (1995, September). <ftp://ftp.loc.gov/pub/z3950/defs/bib1.txt>.
8. Janney, Kody and Sledge, Jane. (1995, September). A User Model for CIMI Z39.50 Application Profile. <http://www.cimi.org/documents/Z3950_app_profile_0995.html>.
9. Library of Congress. (1996, May). Z39.50 Profile for Access to Digital Collections. <http://lcweb.loc.gov/z3950/agency/profiles/collections.html>.
10. The CIMI Profile Release 1.0H. A Z39.50 Profile for Cultural Heritage Information. (1998, November). <http://www.cimi.org/documents/HarmonizedProfile/HarmonProfile1.htm>
11. Weibel, S., Kunze, J., Lagoze, C., Wolf, M. (1998, September). RFC 2413: Dublin Core Metadata for Resource Discovery. <ftp://ftp.isi.edu/in-notes/rfc2413.txt>

П. А. Дортман

ПОДХОДЫ К ОПТИМИЗАЦИИ ПРОГРАММ В СИСТЕМЕ SFP*

1. ВВЕДЕНИЕ

Настоящая работа выполняется в рамках проекта по созданию системы функционального программирования SFP [1]. Эта система позволяет прикладному программисту разрабатывать вычислительные алгоритмы на языке Sisal 3.0 на своей рабочей станции под управлением Windows и генерировать программы для последующего исполнения на суперЭВМ. Система разрабатывается с использованием Microsoft Visual Studio.

Язык Sisal 3.0 — функциональный язык однократного присваивания — был разработан как альтернатива ФОРТРАНу для реализации сложных вычислительных алгоритмов главным образом для научных вычислений [2, 3]. Традиционно для представления Sisal-программ используют представление программы в виде иерархического ациклического потокового графа (например, IF1-представления [4]). В системе SFP внутреннее представление Sisal-программ (будем называть его IR1-представлением) также реализовано как набор классов языка Си++, реализующих средства для работы с IF1-графами.

2. IF1-ПРЕДСТАВЛЕНИЕ

Каждый IF1-граф служит для представления некоторой логически целостной части вычислений, например, для представления в программе функций, отдельных итераций цикла и т.п. А сама программа, таким образом, представляется в виде набора графов функций.

Каждый IF1-граф состоит из *вершин* (операций) и *дуг*, которые представляют собой передачу значения между операциями во время исполнения. Количество различных дуг ограничено, и для каждого типа дуг известно, сколько и какого типа значения вершина принимает. Для удобства упорядоченные места вершин, к которым прикрепляются дуги, будем называть

* Работа выполнена при финансовой поддержке Научной программы «Университеты России» (грант № УР.04.01.027) и Министерства образования РФ (грант № Е02-1.0-42).

входными портами. Места вершин, из которых выходят исходящие дуги с вычисленными значениями, будем называть *выходными портами*. Для вершин каждого типа определено, какие значения на какие входные порты должны приходить, как значения участвуют в вычислении и какие значения формируются на соответствующих выходных портах.

Существуют также специальные константные простые вершины без входных портов, они служат для представления константных значений в вычислениях.

Наряду с простыми вершинами, которые олицетворяют простые операции, такие как арифметические действия, доступ к элементу массива или вызов функции, существуют и составные вершины, состоящие из нескольких подграфов, которые служат для представления циклов и условных конструкций. Так, например, вершина для цикла с предусловием состоит из трех подграфов: граф для представления вычисления условия цикла, граф для вычисления каждой итерации цикла и граф для формирования возвращаемого циклом результата, формируемого по вычисляемым в каждой итерации значениям. Наличие составных вершин в IF1-графах делает их иерархичными.

Ниже приведены примеры графов с простыми и составными вершинами.

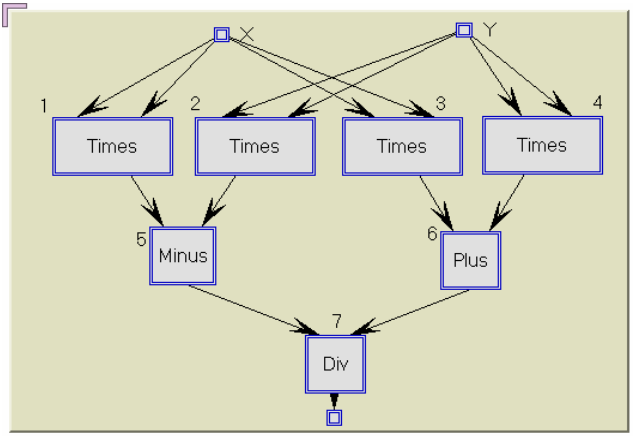


Рис. 1. Граф для вычисления выражения $(x^2 - y^2)/(x^2 + y^2)$

Пример Sisal-функции и его внутреннего представления.

```
function sample(X1, X2, h: real returns real)
 for initial
     X := X1;
     m := 0.0
 while X <= X2 repeat
      X := old X + h;
      m := old m + abs(F(old X) — G(old X))
 returns value of m
end for
end function
```

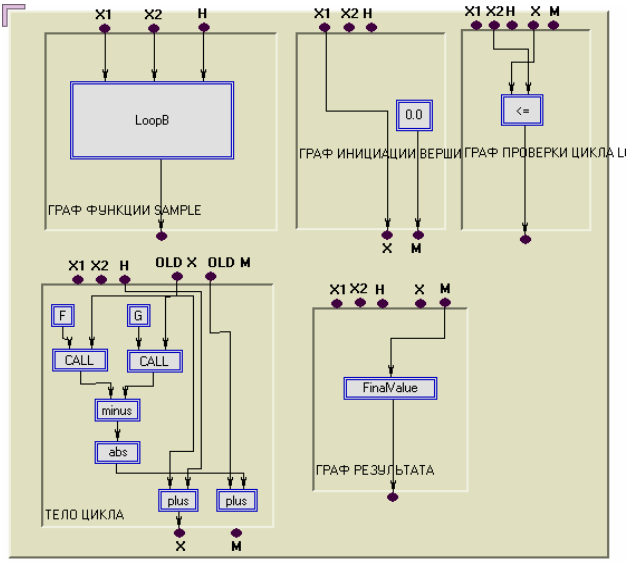


Рис. 2. IF1-граф для примера выше

Большинство традиционных оптимизирующих преобразований над IF1-графами легко программируются просто как преобразования над графами. В настоящей работе мы показываем это на примере ряда преобразований.

Прежде всего отметим, что для ациклического графа, каким является IF1-граф, существует такая нумерация его вершин, которая сохраняет частичный порядок зависимости по данным.

3. ПОДСТАНОВКА ФУНКЦИИ ВМЕСТО ВЫЗОВА

Вызовы функций и процедур зачастую затрудняют автоматический анализ программ в императивных языках. В языке Sisal же вызовы функций не имеют побочных эффектов, их результаты зависят только от своих параметров. В таком случае вызов любой функции может быть заменен кодом с телом этой функции. В IF1-графах подобная подстановка вызова функции на граф тела функции легко алгоритмизируется. И после такой подстановки расширенный граф может быть оптимизирован с помощью других оптимизирующих преобразований, таких как исключение общих подвыражений и вынос инвариантов циклов.

Обычно, когда говорят о подстановке функции, имеют в виду нерекурсивные функции. Однако это вовсе не жесткое правило, раскрытие одного или более рекурсивного вызова может дать дополнительные возможности для проведения других оптимизаций.

4. ИСКЛЮЧЕНИЕ ОБЩИХ ПОДВЫРАЖЕНИЙ

Главной целью преобразования, известного как исключение общих подвыражений, является удаление избыточных вычислений. Отдельные вычисления являются избыточными, если они вычисляются неоднократно при расчете. В IF1-графе избыточные вычисления приобретают форму эквивалентных вершин.

Дадим определение *эквивалентных* вершин. Две простые вершины называются *эквивалентными*, если они представляют одну и ту же операцию (являются вершинами одного типа) и имеют эквивалентные входы. Две составные вершины называются *эквивалентными,* если они являются вершинами одного типа, их входы эквивалентны и их подграфы соответственно изоморфны. Два графа называются изоморфными, если можно установить биективное отображение между их вершинами так, что соответствующие вершины эквивалентны. Будем говорить, что две вершины имеют *эквивалентные входы*, если они имеют одинаковое число входов и для каждой пары входных портов верно одно из следующих утверждений: входные дуги идут от константных вершин с одинаковым значением и типом, входные

дуги идут от одинаковых входных портов границы графа либо входные дуги идут от соответствующих портов эквивалентных вершин.

Понятно, что эквивалентные вершины можно отождествить, так как ведут они себя одинаково, и, таким образом, сократить число вершин-операций в графе. Например, на рис.1 вершины с номерами 1 и 3, а также 2 и 4 — эквивалентны. Исключение общих подвыражений и есть процесс склеивания эквивалентных вершин. Приведем формальный алгоритм исключения общих подвыражений для IF1-графов.

Алгоритм исключения общих подвыражений из графа.
Вход: пронумерованный указанным образом граф G.
Выход: Эквивалентный граф, в котором все эквивалентные вершины склеены.

```
проц ИСКЛЮЧЕНИЕ
    для каждого типа T вершин графа цикл
        Создать пустое множество вершин S(T)
    все
    для каждой вершины N из G в порядке нумерации цикл
        если N — составная то
            для каждого подграфа H из N цикл
                ИСКЛЮЧЕНИЕ(H)
            все
        все
        { поиск эквивалентных вершин }
        пусть S=S(T(N)) — подмножество вершин с таким же типом
                операции T(N), что и у вершины N
        F := 1
        для каждого M из S и  пока F=1 цикл
            если M эквивалентно N то
                Склеить N и M
                F:=0
            все
        все
        если F=1 то S := S+{ N } все
    все
все.
```

Как мы видим, настоящий алгоритм оптимизирующего преобразования графа получился довольно простым и эффективным. Проверка составных вершин теоретически сложна, но на практике не требуется много действий,

чтобы выяснить, что вершины не эквивалентны, так как составные вершины очень отличаются друг от друга.

Некоторые вершины служат для представления коммутативных операций (например сложения или умножения). Однако наш алгоритм не воспринимает, например, 5*G и G*5 как одинаковые выражения. Также настоящий алгоритм не пытается изменить порядок действий для нахождения одинаковых выражений: 5*(G*H) всегда отлично от (5*G)*H.

5. ВЫНОС ИНВАРИАНТОВ ЦИКЛА

Определение выражений, являющихся инвариантом для цикла, и вынос их за цикл, исключают многократное выполнение одного и того же вычисления и являются еще одним традиционным оптимизирующим преобразованием, присущим почти всем оптимизирующим компиляторам. Инвариантом цикла называется выражение, которое вычисляется в одно и то же значение на каждой итерации цикла. Такие выражения могут создаваться программистом с целью повышения читаемости кода или появляться вследствие выполнения каких-либо еще преобразований. Мы сейчас определим, что значит, что IF1-вершина внутри подграфа одной из составных вершин цикла является инвариантом.

Пусть X — вершина внутри подграфа G, содержащегося в графе L цикла. Будем говорить, что X является инвариантом цикла в L, если каждый ее входящий порт принимает значение:

1) либо от константной вершины,

2) либо с порта входных значений L,

3) либо от вершины, являющейся инвариантом цикла.

Предположим, что вершина X является инвариантом цикла или не зависит от каких-либо других вершин, т.е. все ее входящие порты удовлетворяют условиям (1) и (2). В этом случае мы можем вынести X из подграфа G цикла L, последовательно выполняя следующие действия.

1. Вынести X вместе со своими входными дугами из графа G.

2. Поместить вершину X в граф, содержащий L непосредственно перед L в порядке зависимости по данным. Присоединим каждую входную дугу вершины X к соответствующему входному порту L и соединяем выходной дугой исходящие порты с создаваемыми входными портами на L.

3. Соединить вершины, получавшие значения из X, с созданными портами на G.

4. Удалить все входы L, которые использовались исключительно вершиной X.

Понятно, что если мы будем обходить вершины в порядке, сохраняющем зависимость по данным, то проверка условия (3) будет излишней, достаточно проверять только первые два условия. Так как в алгоритме исключения общих подвыражений также использовался обход в порядке, сохраняющем зависимость по данным, то совместное применение этих двух оптимизаций можно естественным образом алгоритмизировать. При этом наиболее удачной нам представляется стратегия, при которой вынос инвариантов цикла выполняется для всех циклов графа, а затем происходит поиск общих подвыражений. Такая стратегия позволяет не упустить шанс склеить только что вынесенные из циклов вершины с другими вершинами в графе.

6. ЗАКЛЮЧЕНИЕ

Семантика IF1-графов позволяет проводить оптимизирующие преобразования программ как эффективные преобразования IF1- графов. И мы продемонстрировали это на примере трех традиционных преобразований.

СПИСОК ЛИТЕРАТУРЫ

1. Глуханков М.П., Дортман П.А., Павлов А.А., Стасенко А.П. Транслирующие компоненты системы функционального программирования SFP // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 69–87
2. Cann D. Retire FORTRAN? A debate rekindled // CACM. —1992. — Vol. 35, N 8. — P. 81–89.
3. Касьянов В.Н., Бирюкова Ю.В., Евстигнеев В.А. Функциональный язык Sisal 3.0 // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 54–67.
4. Густокашина Ю.В., Евстигнеев В.А. IF1 — промежуточное представление Sisal-программ // Проблемы конструирования эффективных и надежных программ. — Новосибирск, 1995. — С. 70–78.

А. А. Дунаев

ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ ИССЛЕДОВАНИЯ БОЛЬШИХ ОДНОМЕРНЫХ МАССИВОВ ДАННЫХ С ПРИМЕНЕНИЕМ КРАТНОМАСШТАБНОГО АНАЛИЗА*

ВВЕДЕНИЕ

При решении ряда научно-исследовательских и практических задач возникает проблема обработки больших массивов данных. В качестве примеров можно привести обработку нуклеотидных последовательностей, глобальное моделирование климата, численные эксперименты в области физики и химии. Основная особенность этих задач заключается в том, что обрабатываемые данные из-за большого объема не могут быть целиком помещены в оперативную память ЭВМ. Так, при обработке нуклеотидной последовательности объем исходных данных имеет порядок 1Гб.

Цель данной статьи — исследовать оптимальные методы работы с большими массивами данных и создать программу, использующую эти методы при обработке данных методом кратномасштабного анализа. Исходными данными для программы является массив данных, сохраненный в файл в том или ином формате. Результаты вычислений сохраняются в файлах и отображаются на экране. Основными требованиями к программе являются по возможности высокая скорость обработки данных и полная функциональность в рамках стандартных средств, предоставляемых операционной системой.

Программа разработана для операционной системы Microsoft Windows NT/2000. Вычислительные блоки, реализующие исследуемые методы, могут быть с минимальными изменениями использованы при разработке других программ.

* Работа выполнена при финансовой поддержке Научной программы «Университеты России» (грант № УР.04.01.027) и Министерства образования РФ (грант № Е02-1.0-42).

1. ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ АЛГОРИТМОВ

1.1. Описание вычислительной задачи

В качестве вычислительной задачи была выбрана реализация метода кратномасштабного анализа. При обработке одномерного массива чисел длины N этим методом (где N является степенью числа 2) получается набор промежуточных массивов, причем каждый последующий массив в два раза короче предыдущего. Общее количество промежуточных массивов равно $\left(\log_2 N\right)-1$. Один из наиболее полных способов визуализации результатов таких вычислений предполагает одновременный параллельный доступ к различным участкам полученных промежуточных массивов. Именно поиск оптимального способа организации такого доступа является основной целью настоящей работы.

1.2. Подготовка исходных данных

В зависимости от конкретного приложения исходные данные могут быть поданы в различных форматах. С другой стороны, для выполнения преобразования наиболее удобным является формат представления данных, при котором отсчеты записаны последовательно в виде чисел с плавающей запятой в двоичном формате. В таком случае становится возможным выполнять вычисления непосредственно после чтения фрагмента файла. Условимся называть файл, содержащий исходные данные, исходным файлом, а его формат — исходным форматом. Внутренний формат представления данных будем называть естественным форматом.

Преобразованные данные записываются в файл в естественном формате, который после исчерпания данных в исходном файле дополняется нулями до оптимального размера, зависящего от конкретного вейвлета, применяемого в данный момент.

1.3. Выполнение вейвлет-преобразования

Для анализа данных используется видоизмененное быстрое вейвлет-преобразование (далее — БВП), опирающееся на метод кратномасштабного анализа, разработанного Малла и Мейером (известного также как пирамидальный алгоритм Малла). Во избежание путаницы будем называть исход-

ный вариант БВП классическим. Классический метод достаточно хорошо изучен и описан [3, 4, 5].

В классическом БВП заложена возможность восстановления первоначального вектора из его гладких компонент и деталей. Поскольку в нашем случае не требуется восстанавливать исходный сигнал, а рассматриваются только гладкие компоненты, можно исключить из рассмотрения компоненты матрицы преобразования, вычисляющие детали. В этом заключается первое отличие применяемого метода от классического БВП.

Второе отличие заключается в длине массива данных, обрабатываемого на отдельном шаге алгоритма. Классическое БВП выполняет свертывание периодического сигнала. Для устранения краевых эффектов последние ряды матрицы преобразования классического БВП выполняют своего рода перенос данных из начала исходного вектора. С другой стороны, в случае, когда сигнал не является периодическим, такой перенос, напротив, вносит нежелательные краевые эффекты. Например, в случае, когда $x_0^0 \neq 0$ и $x_{m-k\ldots m-1}^0 = 0$, причем $0 < k < m-1$, мы получим в первом же векторе гладких компонент $x_{\frac{m}{2}-1}^1 \neq 0$. Для такого «непериодического» преобразования длина исходного вектора $Len(x^0)$ данных должна быть равна одному из членов последовательности:

$$l_n : l_1 = 4, l_k = (l_{x-1}+1)\cdot 2, k \notin Z, k >= 2 .$$

В случае, если

$$\forall k : l_x < Len(x^0) < l_{k+1},$$

т. е. длина исходного вектора попадает между двумя допустимыми величинами, мы расширяем исходный вектор до большего допустимого значения:

$$\tilde{x}^0 : \tilde{x}_a^0 = \begin{cases} x_a^0, a \notin Z, 0 <= a < Len(x^0) \\ 0, a \notin Z, Len(x^0) <= a < l_{k+1} \end{cases}.$$

В конечном итоге применяемый алгоритм использует матрицу преобразования, имеющую следующий вид, и действует на расширенный вектор данных.

$$T = \left[\begin{array}{cccccc:c:cccc} c_0 & c_1 & c_2 & c_3 & 0 & 0 & & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & & 0 & 0 & 0 & 0 \\ 0 & 0 & c_0 & c_1 & c_2 & c_3 & & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & & 0 & 0 & 0 & 0 \\ \hdashline & & & & & & & & & & \\ \hdashline 0 & 0 & 0 & 0 & 0 & 0 & & c_0 & c_1 & c_2 & c_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & & 0 & 0 & 0 & 0 \end{array}\right]$$

1.4. Хранение данных и стратегии обмена данными

После выполнения преобразования программа должна обеспечить визуализацию результатов. Результаты вычислений помещаются в хранилище, задача которого состоит в минимизации задержек в передаче запрашиваемых данных. Внешний программный модуль, получающий данные из массива, управляемого хранилищем, запрашивает блок данных d размера S_d со смещением O_d. В хранилище организуется специальный промежуточный буфер b размера S_b со смещением O_b. Значения S_b и O_b и их пересчет при запросе следующего блока определяются выбранной стратегией предварительной выборки[1]. Для каждой стратегии оптимальными считаются параметры, обеспечивающие минимальное количество операций чтения из файла и минимальный расход памяти.

Были исследованы несколько различных стратегий.

Стратегия 1. При первом запросе блока данных выделяется буфер размером $S_b = k \cdot S_d, k \in Z, k > 3$, после чего в буфер считываются данные из файла со смещением $O_b = O_d - \left(\frac{S_d - S_b}{2}\right)$. При последующих запросах контролируется $R = \min(O_d - O_b, O_b + S_b - (O_d + S_d))$ — расстояние между границами запрашиваемого блока и промежуточного буфера. Чтение нового блока данных из файла производится, когда после запроса очередного блока данных R становится меньше некоторого порогового значения $R_{\min}$, при

[1] Для всех описываемых стратегий размер S_d запрашиваемого блока не меняется от запроса к запросу.

этом смещение O_b выбирается так, что восстанавливается равенство расстояний между границами запрашиваемого блока и буфера: $O_d - O_b = O_b + S_b - (O_d + S_d)$.

Пусть O_d^1 и O_d^2 — смещения блока данных, переданные с двумя следующими друг за другом запросами. Положим $C = O_d^2 - O_d^1$ — относительное изменение смещения для следующего запроса. Оптимальные значения k и $R_{\min}$ можно подобрать, зная $\overline{C}$ — наиболее статистически вероятное C. Эта величина отражает характер способа работы с данными конкретного приложения, поэтому расчет k и $R_{\min}$ возлагается на приложение, использующее хранилище с такой стратегией.

Стратегия симметрична относительно знака C, что позволяет наиболее эффективно использовать ее в таких схемах работы с данными, в которых положительные и отрицательные значения C встречаются приблизительно равновероятно. В то же время, если схема доступа предполагает серии последовательного извлечения данных (в прямом или обратном направлении), такая стратегия становится менее эффективной, поскольку часть буфера b остается неиспользованной.

Стратегия 2. Аналогично первой стратегии, данные буферизуются в промежуточном буфере размером $S_b = k \cdot S_d, k \in Z, k > 3$, и размер буфера остается неизменным все время работы с файлом. Отличие от первой стратегии заключается в способе пересчета O_b при очередном запросе.

При получении очередного запроса проверяется параметр R. Если R оказывается меньше некоторого порогового значения $R_{\min}$, вычисляется C — изменение смещения по сравнению с предыдущим запросом. В зависимости от знака C смещение O_b выбирается так, что устанавливается «перекос» расстояний между границами запрашиваемого блока и буфера в ту или иную сторону:

$$O_b = O_d + S_b - (O_d + S_d) + D \cdot \mathrm{sgn}(C).$$

Параметр D выбирается в диапазоне от 0 до $(S_d - S_b)$. При $D = 0$ получаем результаты, идентичные достигающимся при применении стратегии 1. При $D = (S_d - S_b)$ получаем вариант, оптимальный для однонаправленного доступа: при изменении знака C необходимо немедленно выполнить чтение блока из файла, что вызывает задержку.

Такая стратегия в сравнении со стратегией 1 позволяет уменьшить количество обращений к файлу при тех же затратах памяти. Наиболее эффективно стратегия работает в схемах доступа, предполагающих чередование серий запросов с разным знаком C.

Дополнительное усовершенствование можно внести вводом зависимости D от величины C, и при больших значениях C делать больший перекос расстояний.

Хотя вторая стратегия позволяет оптимально использовать память, неизменный размер буфера накладывает ограничение на максимальное значение C. Если при очередном запросе C превышает этот порог, становится необходимым немедленное пополнение буфера из файла.

Стратегия 3. Аналогично стратегии 2, при вычислении смещения O_b могут учитываться знак и величина C. Основное отличие состоит в том, что S_b также изменяется в зависимости от величины C. Динамическое изменение размера буфера позволяет уменьшить количество операций чтения из файла за счет увеличения объема используемой памяти.

Модель 1 — «линейная». Размер буфера вычисляется по формуле:

$$S_b = S_0 + k \cdot |C|.$$

Эта модель самая простая. S_0 подбирается опытным путем; следует учитывать, что слишком малый размер буфера приведет к увеличению числа операций чтения, а слишком большой — к неэффективному использованию памяти. В этой модели учитывается только последнее значение C.

Модель 2 — «усредняющая». Вычисляется среднее арифметическое среди C для нескольких последних запросов, затем полученное значение используется аналогично первой модели.

Модель 3 — «прогрессирующая». Для нескольких последних запросов определяется тенденция изменения C, в соответствии с которой вычисляется ожидаемое смещение для следующего запроса и новый размер буфера. В качестве вычисляемого параметра, характеризующего эту тенденцию, можно использовать, например, приращение C для последнего запроса относительно предпоследнего.

2. ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ ВЫЧИСЛЕНИЙ

Как уже было отмечено выше, результатом вычислений являются несколько векторов, являющихся приближениями одного и того же исходного вектора. Применительно к исследованиям нуклеотидных цепочек, существует несколько методов визуализации информации подобного рода [6]. В настоящей работе был выбран наиболее наглядный, с точки зрения автора, способ, который заключается в следующем.

Среди всех значений, содержащихся в полученных массивах, выбирается минимальное и максимальное значения. После этого строится цветовая шкала соответствия значения оттенку цвета H в системе цветовых координат HSV. Минимальному значению соответствует цвет с оттенком 0, максимальному — с оттенком 360. После этого массивы отображаются на плоскости рядами цветных точек; цвет точки соответствует значению элемента массива. Такой способ отображения дает возможность визуально выделять характерные участки в массиве данных (рис. 1, 2).

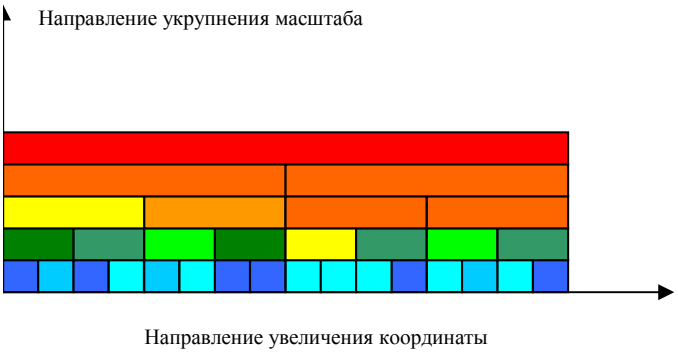


Рис. 1. Визуализация результатов вычислений.
Показан пример графика в увеличенном виде для вектора из 16 элементов.
Каждый ряд точек соответствует масштабу преобразования.
Нижний ряд — исходный вектор x^0.
Точка графика соответствует элементу вектора.
Второй снизу ряд — первый уровень преобразования (вектор x^1), и так далее до x^4

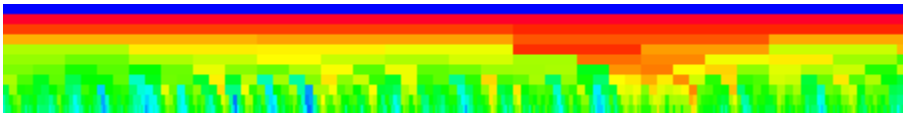


Рис. 2. Показан фрагмент графика для длинного вектора.
График вытянут по оси масштаба для облегчения визуального выделения нужного уровня

3. РЕЗУЛЬТАТЫ

3.1. Практическое применение

Программа использовалась при распознавании экзон-интронной структуры ДНК. Последовательность ДНК, состоящая из элементов A, C, G, T, называемых нуклеотидами, была преобразована к n-мерному вектору.

Возможны следующие варианты преобразования.

1. Метод простого сопоставления [8]. На отрезке **[0,1]** выбираются 4 точки A, C, G, T и сопоставляются соответствующим нуклеотидам. Этот метод использовался в программе.

2. Метод сайт-потенциала [9]. Существует ряд методов [10], [11], позволяющих каждому участку ДНК фиксированной длины сопоставить число в диапазоне **[0,1],** характеризующее потенциал сайта[2] на данном участке. Потенциал характеризует вероятность того, что с этим сайтом будет связываться определенный белок. Таким образом, исходная последовательность может быть преобразована к аналогичному вектору, как и в первом пункте.

Далее применение кратномасштабного анализа позволяет на основе анализа уже известных генов выявить участки, сходные по структуре с генами,

[2] Здесь сайт — потенциальный сайт связывания транскрипционного фактора. Для раскрытия механизмов регуляции экспрессии генов интенсивно проводятся исследования как экспериментального, так и теоретического характера. Одним из базовых понятий, занимающих ключевую роль в процессах транскрипции, являются транскрипционные факторы, которые представляют собой регуляторные белки, обладающие способностью распознавания специфических коротких участков ДНК. Поэтому детальному изучению и распознаванию соответствующих нуклеотидных фрагментов, называемых цис-элементами или сайтами связывания танскрипционных факторов (ССТФ или сайтами), отводится большое внимание.

что дает реальную альтернативу существующим на данный момент методам предсказания генов.

3.2. Выводы

В процессе разработки программного комплекса были исследованы, с одной стороны, различные способы обработки больших массивов данных, стратегии предварительной выборки и алгоритмы сжатия; с другой — возможности применения на практике метода кратномасштабного анализа. Соответственно, результаты работы можно разделить на две части.

Во-первых, разработан программный модуль, позволяющий оптимальным образом организовать работу с большими массивами данных. Реализована возможность выбора различных стратегий предварительной выборки, что позволяет подобрать наилучшую стратегию для каждого конкретного случая.

Во-вторых, построен программный комплекс, использующий описанный модуль для доступа к данным при выполнении кратномасштабного анализа больших массивов данных. Проведена опытная эксплуатация программы. Применение разработанного комплекса дает альтернативу существующим на данный момент методам предсказания генов.

СПИСОК ЛИТЕРАТУРЫ

1. Mowry T. C., Demke A. K. and Krieger O. Automatic Compiler-Inserted I/O Prefetching for Out-of-Core Applications // Proc. of the USENIX 2nd Symp. on OS Design and Implementation (OSDI '96)
2. Kimbrel T., Tomkins A, Patterson R. H. et al. A Trace-Driven Comparison of Algorithms for Parallel Prefetching and Caching // Proc. of the USENIX 2nd Symp. on OS Design and Implementation (OSDI '96)
3. Дремин И. М., Иванов О. В., Нечитайло В. А. Вейвлеты и их использование // Успехи физических наук. — 2001. — Т. 171, № 5.
4. Воробьев В. И., Грибунин В. Г. Теория и практика вейвлет-преобразования. — ВУС, 1999.
5. Астафьева Н. М. Вейвлет-анализ: основы теории и примеры применения // Успехи физических наук. — 1996. — Т. 166, № 11.
6. Дунаев А. А., Кель А. Э., Лобив И. В., Мурзин Ф. А., Половинко О. Н., Черемушкин Е. С. Визуализация генетической информации // Новые информационные технологии в науке и образовании. — Новосибирск, 2003. — С. 147–156.
7. Nelson M. The Data Compression Book. — IDG Books Worldwide, Inc.

8. Cheremushkin E. S., Kel A. E., Lobiv I. V., Murzin F. A., Polovinko O. N. Visualization of DNA sequences by Color Cube Transformation // The 3-d Internat. Conf. on Bioinformatics of Genome Regulation and Structure, 2002.
9. Cheremushkin E. S., Kel A. E. Whole Genome Human/Mouse Phylogenetic Footprinting of Potential Transcription Regulatory Signals // Pacific Symp. on Biocomputing. — 2003. — Vol. 8. — P. 291–302.
10. Kel A. E., Kondrakhin Y. V., Kolpakov Ph. A., Kel O. V., Romashenko A. G., Wingender E., Milanesi L., Kolchanov N. A. Computer tool FUNSITE for analysis of eukaryotic regulatory genomic sequences // Proc. of Third Internat. Conf. on Intelligent Systems in Molecular Biology. — 1995. — P.197–205.
11. Kel AE, Gosling E, Reuter I, Cheremushkin ES, Kel-Margoulis OV, Wingender E. MATCH^{TM}: a tool for searching transcription factor binding sites in DNA sequences // Nucleic Acids Research. — 2003. — Vol. 31, No. 13. — P. 1–4.

В. А. Евстигнеев

МНОГОЧЛЕНЫ ЭРХАРТА*

ВВЕДЕНИЕ

Исследование различных свойств программ и преобразований часто сводится к определению числа целочисленных точек (т.е. точек с целочислеными координатами) в многогранниках (в частности, в многоугольниках). К такой задаче, например, сводится определение мощности множества итераций гнезда циклов, заданного в виде многогранника, в том числе и *параметризованного многогранника*, в котором число целочисленных точек есть функция от этих параметров. Такие функции называются *энумераторами* многогранника.

В 50-е годы прошлого столетия французский математик Эжен Эрхарт (Eugène Ehrhart) обнаружил, что некоторое расширение полиномов, которые потом стали называть *полиномами Эрхарта*, даёт возможность описать энумератор любого многогранника. При этом некоторые свойства таких полиномов находятся в непосредственной связи со свойствами многогранника.

В настоящей статье излагаются основы теории полиномов Эрхарта, следуя работе [1].

1. ПРЕДВАРИТЕЛЬНЫЕ СВЕДЕНИЯ

1.1. Периодические числа

Определим вначале *периодические числа*, которые будут служить коэффициентами для *псевдополиномов*.

Определение 1. *Одномерное периодическое число* есть выражение

$$u(n) = [u_0, \ldots, u_{p-1}] := u_{n \bmod p} = \begin{cases} u_0, \text{ если } n \bmod p = 0, \\ \vdots \\ u_{p-1}, \text{ если } n \bmod p = p-1 \end{cases},$$

*Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (гранты № 02-07-90409 и 04-07-90441).

равное тому значению, индекс которого равен $n \bmod p$. При этом p называется *периодом* для $u(n)$.

Определение 2. *q-мерное периодическое число* $u(n_1, \ldots, n_q)$ определяется через q-мерное поле U величины $p_1 \times \cdots \times p_q$:

$$u(n_1, \ldots, n_q) := U_{(n_1 \bmod p_1, \ldots, n_q \bmod p_q)}.$$

При этом q есть *размерность* поля U. Вектор $p = (p_1, \ldots, p_q)$ называется *мультипериодом* u. Наименьшее общее кратное чисел p_i называется *периодом* u. i-й коэффициент мультипериода называется *периодом размерности* i.

Двумерные периодические числа могут изображаться матрицами. Для более высоких размерностей используется способ представления, известный как "вектор векторов". Примером периодического числа служит

$$[[[0,1]_k, [2,3]_k]_l, [[4,5]_k, [6,7]_k]_l, [[8,9]_k, [10,11]_k]_l]_m.$$

Его размерность равна 3, а его мультипериод равен $\{2,2,3\}$. Следовательно, его период — 6; период размерности 1 есть 2 и т.д.

Период периодического числа может быть увеличен многократно. При этом компоненты соответствующим образом сцепляются. Это же справедливо для периода размерности i многомерного периодического числа. Например, период размерности 2 двумерного периодического числа $[[1,2,3]_k, [3,4,5]_k]_l$ может быть удвоен:

$$[[1,2,3]_k, [3,4,5]_k, [1,2,3]_k, [3,4,5]_k]_l.$$

Значение числа при этом не изменится. Соответственно, периодическое число может быть *редуцировано*, если составляющие элементы в размерности повторяются.

Различные периодические числа одинаковой размерности могут складываться. Пусть p_i и q_i — периоды размерности i двух чисел. Периоды должны быть заменены на наименьшее общее кратное $kgV(p_i, q_i)$, после чего числа складываются покомпонентно. Например, нам нужно сложить два числа $[2,3]_n$ и $[4,5,6]_n$. Наименьшее общее кратное периодов равно 6; следовательно, числа преобразуются к виду $[2,3,2,3,2,3]_n$ и $[4,5,6,4,5,6]_n$. Результатом будет число $[6,8,8,7,7,9]_n$. Периодические числа могут также покомпонентно перемножаться.

Определение 3. *Псевдополином* есть функция $f:\ Z^q \to Z$

$$(n_1,\ldots,n_q) \mapsto \sum_{i_1=0}^{k} \sum_{i_2=0}^{k-i_1} \cdots \sum_{i_q=0}^{k-i_1-\cdots-i_{q-1}} c_{i_1,\ldots,i_q} n_1^{i_1} \cdots n_q^{i_q},$$

где $c_{i_1,\ldots,i_q}$ — коэффициенты периодического числа, наибольшая размерность которого равна q.

Псевдополином характеризуется размерностью, степенью и псевдопериодом. *Размерность* равна q, *степень* есть наибольший показатель k, и *псевдопериод* есть наименьшее общее кратное периодов коэффициентов.

1.2. Многогранники

Введём ряд определений, касающихся многогранников и необходимых в дальнейшем.

Определение 4. *Полиэдром* называется пересечение конечного числа полупространств. Ограниченный полиэдр называется *многогранником* (*политопом*). *Размерностью* полиэдра $\mathcal{P}$ называется размерность аффинного подпространства, целиком содержащего $\mathcal{P}$.

Как альтернатива может быть использовано определение многогранника как выпуклой оболочки конечного множества точек.

Когда речь идёт о двух- или трёхмерных полиэдрах, то интуитивно ясно, что подразумевается под "вершинами" или "рёбрами". Следующее определение формализует эти понятия и обобщает их на полиэдры любой размерности.

Определение 5. Пусть $\mathcal{P} \subseteq Q^n$ — полиэдр и $\mathcal{H} \subseteq Q^n$ — гиперплоскость. Пусть далее $\mathcal{F} := \mathcal{P} \cap \mathcal{H}^+$. Если $\mathcal{F} \neq \emptyset \wedge \mathcal{F} \subseteq \mathcal{H}$, то $\mathcal{H}$ называется *опорной гиперплоскостью*, а $\mathcal{F}$ — *гранью* многогранника $\mathcal{P}$. Каждая грань в $\mathcal{P}$ — снова многогранник.

Грань размерности 0 называется *вершиной*, размерности 1 — *ребром*, размерности $dim(\mathcal{P}) - 1$ — *собственно гранью*.

Если отдельные неравенства, которые описывают различные полупространства, собраны вместе и их коэффициенты сведены в матрицу, то полиэдр может быть представлен в следующем виде:

$$\mathcal{P} = \{\vec{x} |\ A\vec{x} + \vec{b} \geq 0\}.$$

Прежде чем приступить к выводу компактной формулы для подсчёта числа целочисленных точек в полиэдре, определим области допустимости параметров.

Введённые многогранники используются, например, для описания пространства итераций гнезда циклов, причём границы циклов описываются аффинными функциями. Так как эти описания часто зависят от имеющихся в программе параметров, то введём в рассмотрение *вектор параметров* $\vec{n}$, символически охватывающий все имеющиеся параметры.

С помощью этого вектора можно будет параметрически описывать полиэдры. Коэффициенты матрицы A при этом остаются неизменными, изменяться в зависимости от параметров будет только смещение $\vec{b}$.

Определение 6. *Параметрический полиэдр*, соотнесённый с вектором параметров $\vec{n} = (n_1, \ldots, n_r)$, есть полиэдр формы $\mathcal{P} = \{\vec{x} |\ A\vec{x} + C\vec{n} + \vec{b} \geq 0\}$. *Параметрическое пространство* есть подмножество $\mathcal{D}$ пространства N_0^r, которое содержит только допустимые значения $\vec{n}$.

Ясно, что параметры никогда не принимают отрицательные значения.

В исследуемом классе регулярных программ границы массивов суть константы, а границы циклов зависят от параметров программы.

Определение 7. *Параметрическая константа* f размерности m есть m-мерная параметризованная функция

$$f(\vec{x}) = F\vec{x} + F_n\vec{n} + \vec{f},$$

для которой $F = 0$. Так как значение функции при этом больше не зависит от $\vec{x}$, то применяется следующее обозначение:

$$f = F_n\vec{n} + \vec{f}.$$

1.3. Пример

Пусть дан следующий многогранник с вектором параметров $\vec{n} = (P, Q)$:

$$\begin{aligned}\mathcal{P} &= \left\{ \begin{pmatrix} i \\ j \end{pmatrix} | \begin{pmatrix} 1 & 0 \\ -2 & 0 \\ -1 & 1 \\ 0 & -2 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \vec{n} \geq 0 \right\} = \\ &= \left\{ \begin{pmatrix} i \\ j \end{pmatrix} | \, 0 < i \leq \tfrac{1}{2}P \wedge i \leq j \leq \tfrac{1}{2}Q \right\}.\end{aligned}$$

Конфигурация этого многогранника зависит от параметров P и Q. Возможны два случая (см. рис. 1).

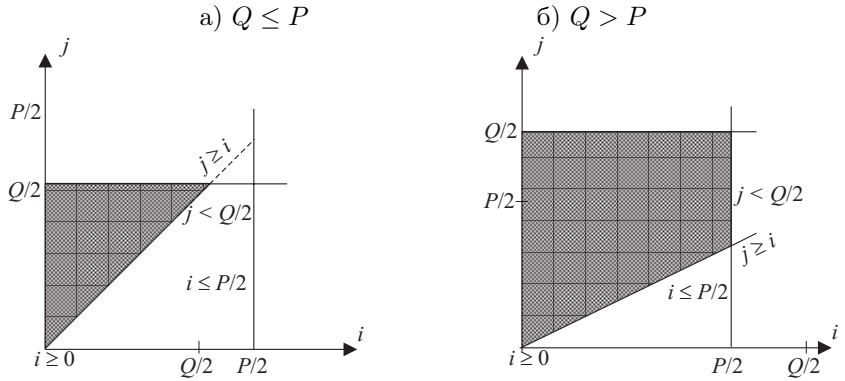


Рис. 1.

2. ЦЕЛОЧИСЛЕННЫЕ ТОЧКИ

Чтобы вывести компактную формулу для подсчёта числа целочисленных точек в многограннике, надо вначале определить область допустимых значений параметров.

Определение 8. *Областью допустимости* параметризованного полиэдра называется максимальное подмножество пространства параметров, для которых множество вершин фиксированно. Пространство параметров $\mathcal{D}$ разлагается на непересекающиеся области допустимости

$$\mathcal{D} = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_n.$$

Множесство вершин для области допустимости $\mathcal{D}_i$ обозначается через $\mathcal{E}_i$.

Определение 9. *Энумератор* параметризованного многогранника в области $\mathcal{D}_i$ есть функция $E : \mathcal{D}_i \mapsto N_0$, которая каждому значению параметра сопоставляет число целочисленных точек в многограннике.

В дальнейшем с понятием многогранника всегда будет связываться некоторая область допустимости параметризованного многогранника.

Определение 10. Многогранник называется *целочисленным*, если все его вершины для всех значений параметров имеют целочисленные коэффициенты. Иначе он — *рациональный*. *Знаменатель* рациональных точек есть наименьшее общее кратное знаменателей их координат. *Знаменатель* рационального многогранника есть наименьшее общее кратное знаменателей его вершин.

Следующее предложение, доказательство которого опускается, утверждает, что знаменатель рационального полиэдра не зависит от параметров как от констант.

Предложение 1. Пусть $\mathcal{P}(\vec{n})$ — параметризованный многогранник. Координаты его параметризованных вершин допускают представление в виде аффинных функций параметров.

2.1. Пример (продолжение)

Для заданного выше многогранника общее пространство параметров имеет вид $\mathcal{D} = \{(P,Q)|\ P \geq 0, Q \geq 0\}$, что означает, что возможные значения параметров ничем больше не ограничены. Чтобы можно было принять во внимание различные формы многогранника, это пространство разбивается на области допустимости. Первая область допустимости есть $\mathcal{D}_1 = \{(P,Q)|\ Q \leq P\} \cap \mathcal{D}$, вторая — $\mathcal{D}_2 = \{(P,Q)|\ Q > P\} \cap \mathcal{D}$. Имеем $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2$.

Для этих областей допустимости множества параметризованных вершин определяются как:

$$\begin{aligned} \mathcal{E}_1 &= \left\{(0,0), \left(\tfrac{Q}{2}, \tfrac{Q}{2}\right), \left(0, \tfrac{Q}{2}\right)\right\}, \\ \mathcal{E}_2 &= \left\{(0,0), \left(\tfrac{P}{2}, \tfrac{P}{2}\right), \left(\tfrac{P}{2}, \tfrac{Q}{2}\right), \left(0, \tfrac{Q}{2}\right)\right\}. \end{aligned}$$

Только две вершины $(0,0)$ и $\left(0, \frac{Q}{2}\right)$ входят в обе области допустимости.

3. ВЫЧИСЛЕНИЕ ПОЛИНОМОВ

Следующее фундаментальное утверждение Эрхарта даёт основу для алгоритма вычисления полиномов.

Предложение 2. Пусть $\vec{n} = (n_1, \ldots, n_p)$ — вектор параметров. Каждый энумератор $E(\vec{n})$ d-многогранника $\mathcal{P}(\vec{n})$ со знаменателем q есть

псевдополином размерности d и степени d. Период его коэффициентов есть максимум q.

Размерность многогранника определяет степень полинома, знаменатель — максимальный период, а число параметров — размерность. Тем самым определяется схематический вид полинома и нужно только определить его коэффициенты.

Знание структуры делает возможным следующий подход к определению многогранника. Нужно придавать параметрам различные конкретные значения и определять число целочисленных точек. Тем самым будут получены значения энумераторов в определенных точках. Отсюда вытекает возможность определять коэффициенты, для чего нужно решить систему уравнений, соответствующих выбранным конкретным значениям.

3.1. Пример (продолжение)

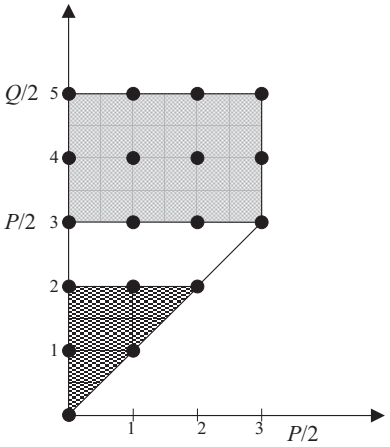
В качестве примера снова возьмём наш многогранник $\mathcal{P}$ с областью допустимости $\mathcal{D}_2$. Размерность многогранника равна 2; кроме того, степень разыскиваемого полинома также равна 2. Знаменатель также равен 2 (см. формулы для величин $\mathcal{E}_1$ и $\mathcal{E}_2$). Отсюда следует, что максимальный период коэффициентов равен 2. Так как число параметров равно 2, размерность полинома должна быть равной 2. Полином теперь имеет следующий вид:

$$\begin{aligned} E_P(P,Q) = {} & [[c_1,c_2]_Q,[c_3,c_4]_Q]_P P^2 + [[c_5,c_6]_Q,[c_7,c_8]_Q]_P Q^2 \\ & +[[c_9,c_{10}]_Q,[c_{11},c_{12}]_Q]_P PQ + [[c_{13},c_{14}]_Q,[c_{15},c_{16}]_Q]_P P \\ & +[[c_{17},c_{18}]_Q,[c_{19},c_{20}]_Q]_P]Q + [[c_{21},c_{22}]_Q,[c_{23},c_{24}]_Q]_P. \end{aligned}$$

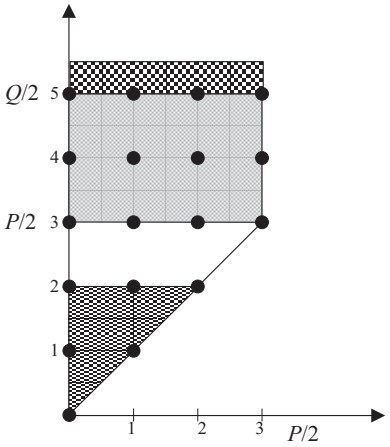
В совокупности нужно определить 24 неизвестных от c_1 до c_{24}. Будем определять периодические числа, исходя из известных четырех частных случаев, указанных на рис. 2. Для случая (а) Q и P — чётные, и мы имеем следующий энумератор:

$$E_P(P,Q) = c_1P^2 + c_5Q^2 + c_9PQ + c_{12}P + c_{17}Q + c_{21}.$$

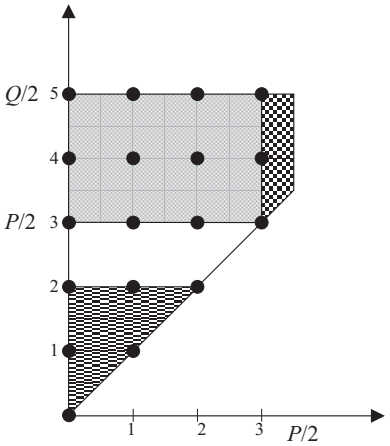
Аналогично получаем ещё три энумератора для оставшихся трёх случаев (б)–(г). Для рассматриваемого случая 6 неизвестных определяются путём подсчёта числа точек для 6 пар конкретных значений для P и Q. Результаты приведены в табл. 1. Из этой таблицы извлекается система из 6 линейных уравнений:



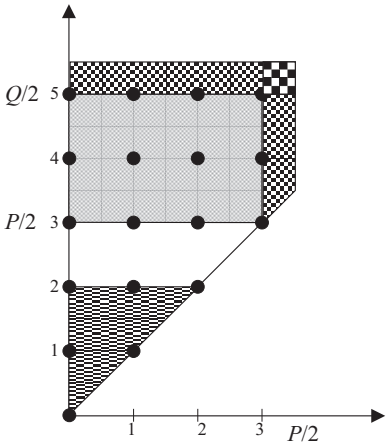
а) P, Q — четные



б) Q нечетно



в) P нечетно



г) P, Q — нечетные

Рис. 2.

Таблица 1

Результаты вычисления многогранника $\mathcal{P}$

P	Q	# точек	$E_P(P,Q)$
0	0	1	c_{21}
0	2	2	$4c_5 + 2c_{17} + c_{21}$
2	2	3	$4c_1 + 4c_5 + 4c_9 + 2c_{13} + 2c_{17} + c_{21}$
2	4	5	$4c_1 + 16c_5 + 8c_9 + 2c_{13} + 4c_{17} + c_{21}$
4	6	9	$16c_1 + 36c_5 + 24c_9 + 4c_{13} + 6c_{17} + c_{21}$
4	8	12	$16c_1 + 64c_5 + 32c_9 + 4c_{13} + 8c_{17} + c_{21}$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 4 & 0 & 0 & 2 & 1 \\ 4 & 4 & 4 & 2 & 2 & 1 \\ 4 & 16 & 8 & 4 & 2 & 1 \\ 16 & 36 & 24 & 4 & 6 & 1 \\ 16 & 64 & 32 & 4 & 8 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_5 \\ c_9 \\ c_{13} \\ c_{17} \\ c_{21} \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 5 \\ 9 \\ 12 \end{pmatrix}.$$

Решение этой системы даёт следующие значения для коэффициентов

$$c_1 = -\frac{1}{8},\ c_5 = 0,\ c_9 = \frac{1}{4}.\ c_{13} = \frac{1}{4},\ c_{17} = \frac{1}{2},\ c_{21} = 1.$$

В остальных трёх случаях также имеем дело с 6 неизвестными. После упрощения получим следующий полином:

$$E(P,Q) = \tfrac{1}{4}QP - \tfrac{1}{8}P^2 + \tfrac{1}{4}\left[[1,0]_Q, [2,1]_Q\right]_P P + \tfrac{1}{4}[2,1]_P Q + \tfrac{1}{8}\left[[1,4]_Q, [5,3]_Q\right]_P.$$

4. ПРИЛОЖЕНИЯ К NUMA-АРХИТЕКТУРАМ

4.1. Введение

Высоко масштабируемые параллельные компьютеры, например SCI-сцепленные кластеры рабочих станций, относятся к NUMA-архитектурам [2]. Таким образом, хорошая статическая локальность существенна для высокой производительности и масштабируемости параллельных программ на этих машинах. В этом разделе, следуя работе [3], описывается

новая техника для оптимизации статической локальности во время компиляции путём применения преобразования и распределения данных. Метрика, с помощью которой оценивается оптимизация, использует полиномы Эрхарта и допускает точное вычисление количества статической локальности. Эффективность новой техники подтверждается экспериментами, которые проводились на SCI-сцепленных кластерах рабочих станций в университете в Падерборне.

Мы используем ограниченную версию блочно-циклической модели распределения, начиная с гнезда параллельных циклов, которые включают аффинные границы циклов и аффинные индексные функции для многомерных массивов. Предполагается, что гнезда содержит только один параллельный цикл. Мы предполагаем, что массивы нарезаны блоками вдоль одной размерности, которые распределены по обрабатывающим узлам. В лучшем случае любой такой блок доступен исключительно для обрабатывающего узла, который владеет блоком. Следовательно, обязанностью преобразований данных является выявление регулярного шаблона блоков, которые доступны из отдельных узлов. Последующее распределение данных должно определить назначение блоков обрабатывающим узлам, которое совместимо с шаблоном. Мы сохраняем распределение вычислений по процессорам, которое было использовано на предыдущих шагах компиляции.

В итоге, для каждого массива в регулярной программе мы автоматически исполняем унимодулярные преобразования данных, которые преобразуют массив, и блочно-циклическое преобразование данных, которое распределяет элементы массива по обрабатывающим узлам. Распределение использует некоторую длину цикла и некоторое смещение.

4.2. Геометрический подход

Мы будем использовать многосеточное приложение из области гидродинамики для иллюстрации нашего подхода. Вычислительное ядро — один из вариантов SOR-цикла.

```
DO I = 2, M+N-2
  FORALL J = MAX(1, I-M+1), MIN(I-1, N-1)
    U(S, I-J, J-1) = (F(S, I-J, J) + U(S, I-J, J-1)
                     + U(S, I-J-1, J) + U(S, I-J, J+1)
                     + U(S, I-J+1,J)/4.0
  ENDFORALL
END DO
```

Это двумерное гнездо циклов с пятью ссылками на массив U и одной ссылкой на массив F. Мы сосредоточимся на ссылках на массив U. Параллельная программа приведённой версии SOR-цикла была получена автоматическим параллелизатором нашего прототипа компилятора. Гнездо циклов показывает параллелизм в самом внутреннем уровне и представляет собой субъект для последующей оптимизации локальности данных.

Теперь мы введём некоторые удобные сокращения и определим фундаментальные геометрические абстракции, приспособленные для классификации преобразований и распределений.

Гнездо циклов N определяет *итерационное пространство* $\mathcal{I}_N$. Массив X, который доступен через ссылку R_l, $l \geq 1$, определяет *индексное пространство* $\mathcal{D}_X$. Через f_l мы сошлёмся на индексную функцию ссылки R_l. Кроме того, P обозначает число процессоров, d — размерность распределения, B — размер блока, j — параллельную размерность. Отныне мы опускаем индексы, если это не вызовет недоразумения. Хорошо известно, что пространство итераций $\mathcal{I}$ и индексное пространство $\mathcal{D}$ оба определяют выпуклые многогранники. Обычно мы представляем выпуклый многогранник $\mathcal{P}$ множеством неравенств, т.е. имеем:

$$\mathcal{P} = \{\vec{x} \in Z^k |\ A \cdot \vec{x} + C \cdot \vec{n} + \vec{b} \geq 0\}.$$

Полином Эрхарта E параметризованного выпуклого многогранника $\mathcal{P}$ есть функция от параметров многогранника, отображающая число целочисленных точек, содержащихся внутри $\mathcal{P}$. Фундаментальная идея нашего подхода — закодировать итерационные точки, которые вызывают локальные доступы через выпуклые многогранники. Тогда полиномы Эрхарта обеспечивают средства для оценки качества преобразования и распределения данных. Мы используем обычную нотацию для полиномов Эрхарта. Обозначение $E(M, N) = [10, 5]_N \cdot M$ есть сокращение, позволяющее различать два случая: $E(M, N) = 10 \cdot M$, если $N \bmod 2 = 0$, и $E(M, N) = 5 \cdot M$, если $N \bmod 2 = 1$, соответственно. Эта вычислительная схема расширяет циклические коэффициенты больших размерностей. Более того, многогранник может иметь множество полиномов Эрхарта. В этом случае его полиномы определяются для выпуклых подмножеств пространства параметров, называемых областями правильности (validity domains).

Теперь наши два примитива для распределения данных — агрегирование блоков (block aggregation) и свёртка блоков (block convolution) —

могут быть оттранслированы в терминах выпуклых многогранников. Мы начнём с примитива, который адресует агрегирование блоков. Он собирает эффект от распределения данных с блоками одинакового размера.

Пусть $\vec{x}' = f_l(\vec{x}) \in \mathcal{D}$ обозначает индексную точку, доступную через ссылку R_l в итерационной точке $\vec{x} \in \mathcal{I}$. Так как распределение данных применяется к размерности d, блок, идентифицируемый как $x'_d = \lfloor x_d/B \rfloor$, доступен в $\vec{x}$. Нелинейное выражение $\lfloor x_d/B \rfloor$ может быть преобразовано в линейное за счёт дополнительной неизвестной b и ограничения C_d. Если мы встретим уравнение, содержащее $\lfloor x_d/B \rfloor$, то заменим его на новую свободную переменную b и дополнительное ограничение на допустимые границы изменения x_d для удовлетворения C_d: $B \cdot b \le x_d \le B \cdot (b+1)$.

Мы продолжим с примитивом для свёртки (конволюции) блоков. Он суммирует эффект циклического распределения данных. Поэтому пусть $b = f_l(\vec{x})$ обозначает выражение, которое вычисляет в блоке, доступном через ссылку R_l в итерационной точке $\vec{x} \in \mathcal{I}_N$. Потом этот блок будет распределён процессору $b \bmod P$. Выражение $b \bmod P$ должно быть преобразовано в линейное выражение для подстановки в наш линейный шаблон. Мы заменяем его на $(b - P \cdot z)$, где z — новая свободная переменная, и дополнительно ограничим выражение b для удовлетворения C_b: $P \cdot z \le b \le P \cdot (b+1)$.

Таким образом, мы можем использовать приведённые выше примитивы для описания множеств итерационных точек, не покидая области выпуклых многогранников.

4.3. Преобразование данных

Предлагаемый метод для выбора преобразований данных может быть разделён на две фазы. Первая фаза вычисляет множество оптимальных преобразований и распределений для каждой ссылки отдельно. Этот подход гарантирует достижение цели в случае инъективных индексных функций, и оптимальность совпадает с отсутствием отдалённых доступов. Вторая фаза упорядочивает эти преобразования-кандидаты; она сравнивает сопоставленные им полиномы Эрхарта, рассматривая все (!) ссылки совместно и выбирая лучшее преобразование из всех преобразований-кандидатов.

На рис. 3 показаны три главных шага в генерации преобразований-кандидатов. В течение первого шага выбираются базисные векторы, ко-

торые стягивают подпространства итерационного пространства так, что эти подпространства доступны только одному процессору (а). Потом эти векторы отображаются в индексное пространство, где они стягивают подпространства, которые доступны уже более чем одному процессору (б). Внутри следующего шага определяется унимодулярное преобразование, которое делает эти подпространства ортогональными одной из осей (в). Полученный массив нарезается на блоки вдоль выбранной оси. Каждый из этих блоков либо не используется, либо используется только одним процессором, что ведёт к некоторому шаблону использования (utilization pattern) блоков памяти. Наконец, определяется смещение для перемещения шаблона так, чтобы он отображал распределение данных. Результатом является преобразование, которое возвращает все (!) доступы, исполняемые одной ссылкой в локальных доступах.

4.3.1. Упорядочение ссылок

Вначале покажем, как упорядочить преобразование данных относительно отдельной ссылки R. Мы начинаем с выпуклого многогранника итерационного пространства $\mathcal{I}$ и разлагаем его на подпространства $\mathcal{I}_p$ так, что подпространство $\mathcal{I}_p$ исполняется на процессоре P_p. Таким образом,

$$\mathcal{I} = \{\vec{x} \in Z^k |\ A \cdot \vec{x} + C \cdot \vec{n} + \vec{b} \geq 0\}$$

для подходящих матриц A, C и вектора $\vec{b}$. Полином Эрхарта I для $\mathcal{I}$ показывает число итераций, которые могут исполняться всеми параллельными процессорами. В случае примера с мультирешётками получаем полином следующего вида:

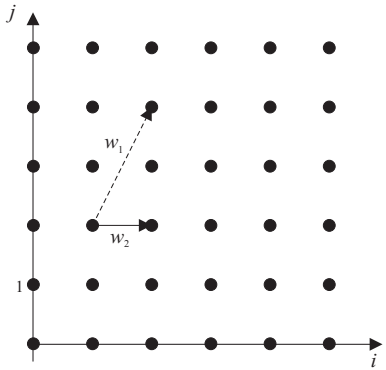
$$I(M, N) = M \cdot N - M - N + 1.$$

Так как мы предположили циклическое отображение итерационных точек в параллельной размерности j в множество из P параллельных процессоров, то получаем, что

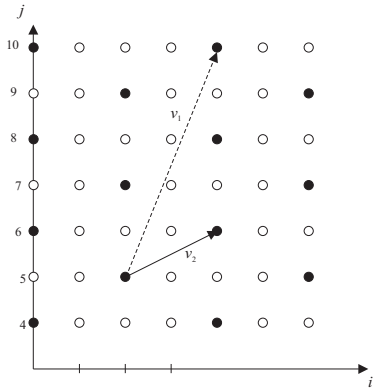
$$\mathcal{I}_p = \{\vec{x} |\ \vec{x} \in I \wedge x_j \bmod P = p\}.$$

Таким образом, каждое множество $\mathcal{I}_p$ также есть выпуклый многогранник, и существует полином Эрхарта I_p для $\mathcal{I}_p$. Например, для рассматриваемого примера

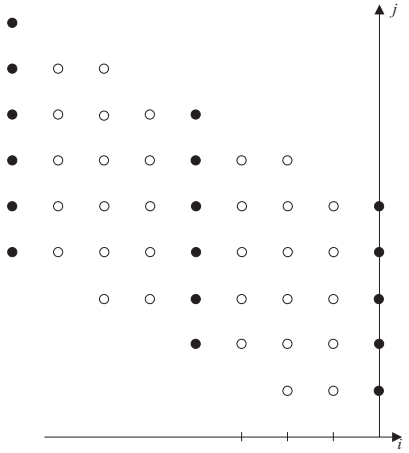
$$I_p(M, N) = \frac{1}{2}(M \cdot N - M - \left[\begin{array}{cc} 2 & 1 \\ 0 & 1 \end{array}\right]_{p,M} \cdot N + \left[\begin{array}{cc} 2 & 1 \\ 0 & 1 \end{array}\right]_{p,M}).$$



а) итерационное пространство



б) индексное пространство



в) новое индексное пространство

Рис. 3.

Чтобы вычислить индексную точку внутри преобразованного индексного пространства, нам нужно применить преобразование

$$\vec{x} \mapsto T \cdot \vec{x} + T_n \cdot \vec{n} + \vec{t}$$

для индексной точки $f(\vec{x})$. Потом мы можем исследовать блочно циклическое распределение массива, для того чтобы выявить, является ли элемент массива $t(f(\vec{x})) = \vec{x}''$, к которому имеется доступ из итерационной точки $\vec{x}$, локальным элементом массива процессора P_p. Согласованное ограничение C_p, таким образом, имеет вид:

$$C_p : \; B \cdot p \leq \pi_d(\vec{x}'') - (B \cdot P) \cdot z_2 < B \cdot (p+1).$$

Заметим, что $\pi_d(\vec{x})$ обозначает проекцию на компоненту x_d. Таким образом, множество итерационных точек $\mathcal{L}_p$, которые порождают доступы к элементам локального массива, равно

$$\mathcal{L}_p = \{\vec{x} \in {}_p |\; C_p(\vec{x} = true\}.$$

Заметим, что множество $\mathcal{L}_p$ — также выпуклый многогранник. Множество $\mathcal{R}_p$, которое порождает удалённые доступы, равно разности $\mathcal{R}_p := \mathcal{I}_p - \mathcal{L}_p$ и в общем случае не является выпуклым многогранником. Тем не менее полином для $\mathcal{R}_p$ существует и даёт число удалённых доступов. Для ссылки $\mathcal{R}_1 = \mathtt{U(I-J, J-1}$ нашего примера и для $B = 1$, $t = id$, получаем:

$$L_0(M,N) = \frac{1}{4}(M \cdot N - [2,1]_N \cdot M - [2,1]_M \cdot N + \left[\begin{array}{cc} 4 & 2 \\ 2 & 1 \end{array}\right]_{N,M}).$$

Отметим, что L_0 есть специализация полинома $L(M,N,p)$ для $p = 0$. Следовательно, мы можем вычислить полином $|L_0(M,N) - L_1(M,N)|$, который будет означать *небаланс* доступов к удаленной памяти для задействованных процессоров. Ниже мы рассмотрим подробнее действие этого небаланса.

4.3.2. Ранжирование преобразований

Покажем, что представленная выше конструкция $\mathcal{L}_p$ позволяет определить отношение локальных (удалённых) доступов любой ссылки $\mathcal{R}_l$. Мы начнём с итерационного пространства как параметризованного многогранника, включаем новый параметр p для отбора итерационных точек, исполняемых процессором P_p, и далее ограничиваем этот многогранник так, чтобы он содержал только точки с локальным доступом.

Если мы опустим параметр p, который представляет процессор P_p, то мы получим желаемый многогранник $\mathcal{L}_l$.

Объём многогранника $\mathcal{L}_l$ представляется полиномом Эрхарта L_l, который служит в качестве метрики для ранжирования преобразований относительно ссылки R_l. Таким образом, сумма $\sum_l L_l$ по всем ссылкам, дополненная полиномом G, который изображает полное число доступов к памяти, отражает отношение локальных (удалённых) доступов исходного гнезда циклов N и обеспечивает желаемую метрику для ранжирования комбинаций линейного преобразования данных и блочно-циклического распределения данных.

Мы не рассматриваем случай кратных областей правильности для многогранников $\mathcal{L}_l$. В этом случае нужно больше информации относительно параметров программы, для того чтобы выбрать правильную область правильности.

4.3.3. Заключительный отбор

Чтобы окончательно выбрать преобразование, которое хорошо реализуется для входного гнезда циклов, мы *символически сравним* полиномы Эрхарта для различных преобразований и сохраним лучшее из всех преобразований-кандидатов. Для заданного конечного множества преобразований, которое будет сконструировано ниже в разд. 4.4, сравним полиномы L этих преобразований. Без дальнейших знаний о параметрах программы мы вначале упростим периодические коэффициенты, т.е. мы заменим их арифметическими средними, унифицируем параметры программы и потом сравним коэффициенты полиномов в нисходящем порядке. Следующие полиномы Эрхарта E_M и E_N суть полиномы для нашего текущего примера для двух параллельных процессоров:

$$E(M,N,p) = \frac{1}{4}\cdot(5\cdot M\cdot N - \left[\begin{array}{cc} 10 & 5 \\ 0 & 5 \end{array}\right]_{p,N}\cdot M - \left[\begin{array}{cc} 6 & 5 \\ 4 & 5 \end{array}\right]_{p,M}\cdot N + \left[\left[\begin{array}{cc} 12 & 10 \\ 6 & 5 \end{array}\right]_{N,M}, \left[\begin{array}{cc} 0 & 0 \\ 4 & 5 \end{array}\right]_{N,M}\right]_p),$$

$$E(M,N,p) = \frac{1}{4}\cdot(6\cdot M\cdot N - \left[\begin{array}{cc} 12 & 6 \\ 0 & 6 \end{array}\right]_{p,N}\cdot M - 6\cdot N + \left[\begin{array}{cc} 12 & 6 \\ 0 & 6 \end{array}\right]_{p,N}).$$

Полином E_M представляет собой дефолт-распределение вдоль оси M, в то время как E_N представляет распределение вдоль оси N. Оба полинома отображают число локальных доступов, откуда следует, что распределение данных вдоль оси N "старше// распределения вдоль оси M, поскольку $\frac{6}{4} \cdot M \cdot N > \frac{5}{4} \cdot M \cdot N$ для значащих параметров M, N задачи. Более того, эти члены не зависят от процессорной координаты p. Распределение массива U вдоль оси M (E_M) показано на рис. 3.

4.4. Перечисление преобразований

Хотя результаты предыдущего подраздела позволяют ранжировать преобразования данных или заданную формулировку программы и поэтому обеспечивают точный результат сами по себе, нас будет интересовать перечисление *преобразований-кандидатов*, которое обеспечивает локальность, для того чтобы выбрать лучшее с помощью метрики L.

Вначале мы выберем $n-1$ векторов $\vec{w}_1, \ldots, \vec{w}_{n-1}$ внутри n-мерного итерационного пространства $\mathcal{I}$, которые стягивают непересекающиеся подпространства размерности $n-1$. Если $\mathcal{I}_{\vec{o}}$ — такое подпространство, идентифицируемое некоторым началом $\vec{o}$, т.е. $\mathcal{I}_{\vec{o}} = \{\vec{x} \in \mathcal{I} |\ \vec{x} = \vec{o} + \sum_{i=1}^{n-1} k_i \cdot \vec{w}_i,\ k_i \in Q\}$, то следующая импликация должна выполняться:

$$\vec{x}, \vec{x}' \in \mathcal{I}_{\vec{o}} \Rightarrow x_p \bmod P = x'_p \bmod P.$$

Таким образом, мы предназначаем подпространство $\mathcal{I}_{\vec{o}}$ отдельному процессору. В терминах порождающих векторов $\vec{w}_i$ мы требуем для произвольных итерационных точек $\vec{x},\ \vec{x}' \in \mathcal{I}$, чтобы

$$\vec{x}' = \vec{x} + \sum_{i=1}^{n-1} (k_i \cdot \vec{w}_i) \Rightarrow x_p \bmod P = x'_p \bmod P.$$

Теорема 4.1 даёт достаточное условие, которое позволяет выбор $\vec{w}_i$. Заметим, что ниже p обозначает параллельную размерность гнезда циклов.

Теорема 4.1. Пусть $\vec{w}_1, \ldots, \vec{w}_{n-1}$ — линейно независимые порождающие векторы из Z^n такие, что $\vec{w}_i = (w_{1,i}, \ldots, w_{n-1,i})^t$. Тогда эти векторы удовлетворяют сформулированному выше ограничению, если:

i) $\forall i:\ \gcd(w_{1,i} \ldots, w_{n-1,i}) = 1$ и

ii) $\forall j$: существует самое большее одно $i:\ w_{j,i} \neq 0$ и

iii)$\forall i:\ w_{p,i} \bmod P = 0$.

Следующая импликация применяется к индексному пространству.

Теорема 4.2. Пусть $\vec{w}_1, \ldots, \vec{w}_{n-1}$ суть линейно независимые векторы из Z^n, удовлетворяющие приведённому выше ограничению. Пусть $f(\vec{x}) = F \cdot \vec{x} + F_n \cdot \vec{n} + \vec{b}$ обозначает индексную функцию, имеющую квадратную обратимую матрицу доступа F. Пусть далее $\vec{v}_i = F \cdot \vec{w}_i$ обозначают образы векторов $\vec{w}$ относительно линейной части индексной функции f. Тогда:

$$f(\vec{x}') = f(\vec{x}) + \sum_{i=1}^{n-1} k_i \cdot \vec{v}_i \Rightarrow x_p \bmod P = x'_p \bmod P.$$

Таким образом, векторы $\vec{v}_i$, упоминаемые в теореме 4.2, стягивают подпространства индексного пространства, которые доступны из самое большее одного процессора. Формулировка теоремы 4.2 влечёт, что включаются только те индексные точки, которые имеют копию в итерационном пространстве. Рис. 3(а) иллюстрирует порождающие векторы $\vec{w}_i$, в то время как рис. 3(б) иллюстрирует векторы $\vec{v}_i$ для обеих теорем.

Остаётся вычислить преобразование T. Пусть $\vec{v}'_i = T \cdot \vec{v}_i$ обозначает образ $\vec{v}_i$ при преобразовании T. Если существует индекс j такой, что все векторы $\vec{v}'_i$ имеют 0-позицию в их j-й компоненте, тогда достаточно распределить индексные точки вдоль этой размерности. Мы вычисляем такое преобразование T, применяя гауссовское исключение, комбинируя векторы $\vec{v}_i$ в матрицу из n строк и $n-1$ столбцов. Её ранг равен $n-1$, потому что векторы $\vec{v}_i$ линейно независимы. Теперь мы исключаем элементы последней строки гауссовским исключением и замещаем эту строку на желаемом уровне. Алгоритм исключения даёт нам преобразование T.

4.5. Распределение данных

Пока мы вычислили матрицу преобразования. Остаётся определить параметры распределения. Это делается в два этапа. Вначале мы определяем окончательный шаблон использования (utilization pattern), а потом будет вычислено смещение для функции преобразования.

4.5.1. Шаблон использования

Вначале мы введём важное понятие вырезки массива (array slice).

Определение 11. *Вырезкой* массива X относительно размерности d называется подпространство индексного пространства $\mathcal{D}_X$, которое есть результат оценки фиксированной координаты в размерности d. Говорят, что процессор *владеет* вырезкой S_k, если он имеет доступ к элементам внутри вырезки, но никакой другой процессор такого доступа не имеет.

Шаблон пользования состоит из вырезок, которыми владеет специфицированный процессор, и из неиспользуемых вырезок. Используя $'*'$ для обозначения неиспользуемых вырезок, мы можем описать шаблон для преобразований-кандидатов на рис. 3(в) как $'0, *, *, *, 1, *, *, *'$. Имеем вырезку, принадлежащую процессору 0, за ней следуют три неиспользуемые вырезки, затем вырезка, принадлежащая процессору 1 и т.д. Этот шаблон повторяется циклически. Блоки с неиспользуемыми вырезками всегда имеют один и тот же размер: в нашем случае 3.

Поэтому для вычисления этого шаблона может быть использован простой итеративный алгоритм. Должны быть выделены три случая. В первом случае шаблон непосредственно встраивается в циклическое разбиение. Во втором случае к массиву должно быть применено обратное преобразование, чтобы сделать возможным включение шаблона в распределение, что, например, верно для шаблона $'2, *, 1, *, 0, *'$. В третьем случае шаблон не может быть встроен в распределение. Следовательно, эти пребразования удаляются из множества законных кандидатов.

4.5.2. Смещение

Вплоть до данного момента мы точно знали порцию T преобразования $t(\vec{x}) = T \cdot \vec{x} + T_n \cdot \vec{n} + \vec{t}$. Смещение $T_n \cdot \vec{n} + \vec{t}$ должно быть выбрано так, чтобы каждый процессор имел доступ только к тем вырезкам, которыми он сам владеет. Это свойство удовлетворяется для индексной функции без смещения. Мы должны определить смещение преобразования t так, чтобы оно компенсировало бы смещение f.

В контексте нашего простого процессорного отображения итерационная точка $\vec{0}$ исполняется на процессоре 0. Более того, вырезка S_0 всегда находится во владении процессора 0. Таким образом, достаточно выбрать смещение так, чтобы итерационная точка $\vec{0}$ вызывала бы доступ к вырезке S_0. Начиная с $t(f(\vec{0})) = \vec{0}$, мы получаем $T_n = -T \cdot F_n \wedge \vec{t} = -T \cdot \vec{f}$.

5. ЗАКЛЮЧЕНИЕ

Из экспериментов следует, что техника, основанная на полиномах Эрхарта, значительно улучшает обработку регулярных программ на NUMA-архитектурах. Она годится для улучшения распределения данных в программах с явным параллелизмом и для руководства оптимизацией распределения данных в распараллеливающих компиляторах.

СПИСОК ЛИТЕРАТУРЫ

1. Heine F. Optimierung der Datenverteilung für SCI-gekoppelte Workstation-Cluster. Master's thesis, Universität-GH Padeborn, May 1999.
2. Евстигнеев В.А., Мирзуитова И.Л. Развитие NUMA-архитектуры: текущее состояние // Современные проблемы конструирования программ. — Новосибирск, 2002. —- С. 139–154.
3. Heine F., Slowik A. Volume driven data distribution for NUMA-machines // Lect. Notes Comp. Sci. — Vol. 1900. — 2000. — P. 415 – 424.

В. Н. Касьянов, Е. В. Касьянова

ДИСТАНЦИОННОЕ ОБУЧЕНИЕ: МЕТОДЫ И СРЕДСТВА АДАПТИВНОЙ ГИПЕРМЕДИА*

ВВЕДЕНИЕ

В недавнем прошлом хороший почерк был гарантией спокойной и обеспеченной жизни до старости. Последние десятилетия характерны ускорением обновляемости технологий и знаний в различных сферах деятельности человека. Поэтому школьного и даже вузовского образования надолго уже не хватает. Сегодня особенно актуальна концепция непрерывного образования на протяжении всей жизни или, как говорят, пожизненного обучения (long-life education). Так, на последнем в XX веке 16-ом Всемирном компьютерном конгрессе IFIP, который проходил в Пекине в 2000 г. под лозунгом «Обработка информации. За рубежом 2000», из 132 докладов по использованию информационных и коммуникационных технологий в обучении 38 докладов было посвящено пожизненному обучению, 55 — подготовке преподавателей и лишь 39 — обучению информатике [4].

В разных странах создаются специализированные открытые университеты: например, Каталонский открытый университет, Британский открытый университет и др. В Европе и Северной Америке создаются консорциумы ведущих университетов, предоставляющих широкий спектр дистанционных образовательных услуг. Так, ассоциация дистанционного обучения в США объединяет в своем составе пять тысяч учебных заведений. ЮНЕСКО (UNESCO) ведет работу по организации распределенного университета, обучение в котором будет происходить в виртуальном пространстве, вне зависимости от расселения и границ и без ограничений по времени.

Поиск соответствующей организационной структуры и учреждений образования (особенно образования взрослых), которые обеспечили бы переход от принципа «образование на всю жизнь» к принципу «образование через всю жизнь» — важнейшая проблема XXI века.

Открытое образование — это образование, доступное всем. Его развитие неизбежно приведет к существенному пересмотру традиционных мето-

* Работа выполнена при финансовой поддержке Министерства образования РФ (грант № E02-1.0-42).

дик и технологий учебного процесса, а также к формированию единого открытого образовательного пространства.

Книги, а в более ранние времена — папирус, телячья кожа или каменные плиты, были излюбленными носителями информации со времени изобретения письменности. Хранение информации в виде книг имеет множество преимуществ: информация сгруппирована воедино, и зачастую одна тема исследуется от начала до конца.

Вы можете учиться, читая книги или дискутируя с другим человеком. Основное различие этих двух подходов состоит в том, что партнер-человек принимает во внимание особенности конкретного ученика и адаптирует скорость подачи и глубину информации к способностям своего визави. Обычная книга не в состоянии адаптировать себя к текущим потребностям читателей — однажды напечатанная, она остается статичной. Если представить себе книгу, написанную специально для нас, она наверняка будет обладать множеством достоинств. Например, нам не придется скучать, читая то, что нам уже известно или вовсе неинтересно. Представьте учебник, иллюстрирующий сложные предметы примерами из области вашего главного хобби, учебник, дающий полное и серьезное изложение определенной темы и всегда — даже через много лет — ссылающийся на самые последние исследования по этой теме. Вообразите учебник, всегда привлекающий к себе внимание, потому что в нем в любой момент есть новая, интересная и полезная вам информация и пояснения, специально подобранные к вашему уровню знаний.

С момента появления Всемирной Сети (WWW) в 1991 г. [10] информация приобрела новое измерение. Сегодня миллионы компьютеров соединены через Интернет, и их пользователи могут получить информацию из любой точки мира.

Такое огромное количество информации дает возможность обучения и приобретения опыта. Но эффективное извлечение информации из Интернета все еще остается важной темой для исследований, так как эффективность поисковых машин возрастает с увеличением точности запроса. Информация из Интернета часто остается бесполезной для целей обучения, поскольку обучающийся нуждается в руководстве для построения ментальной модели области, в которой он работает, прежде чем он сможет задавать достаточно точные запросы.

Было бы очень полезным иметь разные учебники для разных типов учеников, для студентов с различными интересами и различным уровнем начальной подготовки в изучаемой области. Делая шаг в этом направлении, адаптивные гиперкниги персонализируют свое содержание применительно

к конкретным потребностям пользователей. Они дают пользователю способность самим определять цели своего обучения, предлагают разумные последующие шаги, поддерживают обучение, основанное на проектах, дают альтернативный взгляд и могут быть дополнены документами, написанными самими обучающимися. Адаптивные гиперкниги являются информационными репозиториями с доступом к распределенной информации. Реализованные в качестве Интернет-приложений, они могут интегрировать документы, расположенные где угодно в Интернете, и адаптировать эти документы к целям и способностям конкретного пользователя.

Многообещающие результаты дают исследования в области адаптивных гипермедиа-систем [17], которые объединяют гипермедиа-системы с интеллектуальными обучающими программами для адаптации системы к потребностям конкретного пользователя.

В данной работе рассматриваются вопросы поддержки дистанционного обучения, особое внимание уделяется анализу методов и средств адаптивной гипермедиа, используемых современными адаптивными обучающими Web-системами.

Статья начинается с описания свойств пяти поколений систем дистанционного обучения, начиная от систем обучения по переписке, больше известных в СССР как системы заочного обучения, и кончая системами гибкого обучения и системами интеллектуального гибкого обучения, определяющих настоящее и будущее дистанционного образования и базирующихся на Web-технологиях (разд. 1). В разд. 2 определяются адаптивные гипермедиа-системы, рассматриваются их основные типы и области применения. Разд. 3 отвечает на вопрос, к чему адаптируются адаптивные обучающие системы; здесь изучаются такие характеристики пользователя, важные для адаптации, как цели пользователя, его знания, уровень подготовки пользователя и имеющийся у него опыт, предпочтения и личностные характеристики пользователя. Вопросу, что адаптируется в системах Web-обучения, посвящен разд. 4; в нем рассматриваются такие технологии, как выстраивание программы обучения, интеллектуальный анализ решений обучаемого, поддержка в решении задач, привнесенных в обучающие Web-системы из несетевых интеллектуальных обучающих систем, а также технологии адаптации содержания и навигации, характерные для адаптивных гипермедиа-систем. Моделям предметной области, пользователя и адаптации, используемым в адаптивных гипермедиа-системах, посвящен разд. 5. Разд. 6 содержит примеры существующих экспериментальных адаптивных обучающих Web-систем. Завершает изложение заключение, в котором объясняется, почему среди коммерческих обучающих Web-систем мало адаптивных.

1. ПЯТЬ ПОКОЛЕНИЙ СИСТЕМ ДИСТАНЦИОННОГО ОБУЧЕНИЯ

Многие годы университеты, активно работающие в области дистанционного и открытого образования, находились и находятся на передовых позициях в использовании новых технологий для увеличения доступа к возможностям обучения и подготовки. Модели дистанционного обучения, используемые ими, постепенно эволюционировали, пройдя через следующие четыре стадии [88]: модель обучения по переписке, мультимедийная модель обучения, модель телеобучения, модель гибкого обучения.

Модель обучения по переписке поддерживается системами дистанционного обучения первого поколения, которые основываются на печатной технологии и являются гибкими с точки зрения времени, места и пространства. Эти системы не обладают высокой интерактивностью, но предоставляют хорошо проработанный учебный материал.

Мультимедийная модель характеризуется системами дистанционного обучения второго поколения, которые, сохраняя гибкость с точки зрения времени, места и пространства и хорошую проработанность учебного материала, используют в качестве технологий доставки не только печатную продукцию, но и аудиокассеты и видеозаписи, а также такие высоко интерактивные средства доставки, как компьютерное обучение и интерактивное видео.

Модель телеобучения связана с системами дистанционного обучения третьего поколения, которые базируются на следующих технологиях: аудиоконференция, видеоконференция, широковещательное ТВ/радио и аудиовидеоконференции. Эти системы привязаны к определенному времени, месту и пространству, обладают высокой интерактивностью, но в случае с аудиовидеоконференциями могут поставлять не вполне проработанный учебный материал.

Модель гибкого обучения поддерживается системами дистанционного обучения четвертого поколения, которые используют в качестве технологий доставки интерактивный мультимедийный диалог, основанный на Интернете, доступ к WWW-ресурсам и компьютерное взаимодействие. Эти системы обладают высокой интерактивностью и предоставляют хорошо проработанный учебный материал.

И хотя в последние годы многие университеты только начинают разворачивать системы дистанционного обучения четвертого поколения, эволюционный процесс продолжается, и на основе дальнейшего использования новых технологий уже начинают появляться системы дистанционного обу-

чения пятого поколения, поддерживающие так называемую *модель интеллектуального гибкого обучения* [88].

Системы дистанционного обучения пятого поколения используют для доставки следующие технологии: интерактивный мультимедийный диалог, основанный на Интернете; доступ к WWW-ресурсам; компьютерное взаимодействие на основе автоматизированных ответных систем; доступ через портал кампуса к процессам и ресурсам учреждения. Эти системы обладают высокой интерактивностью и предоставляют хорошо проработанный учебный материал.

Следует отметить, что системы первых трех поколений, а также системы четвертого поколения, опирающиеся на компьютерное взаимодействие, характеризуются переменной стоимостью, которая имеет тенденцию к увеличению или уменьшению, напрямую связанную (часто линейно) с изменением объема активности; например, в системах доставки второго поколения распространение пакетов материалов для самостоятельного обучения имеет переменную стоимость, которая изменяется прямо пропорционально числу обучаемых студентов. В отличие от таких систем, системы дистанционного обучения пятого поколения обладают тенденцией к значительному уменьшению стоимостей, связанных с обеспечением доступа к учрежденческим процессам и онлайновому обучению отдельных пользователей.

Центральной составляющей систем пятого поколения является разработка принимаемого е-интерфейса, портала кампуса, через который студенты, преподаватели и другие штатные сотрудники могут связываться с университетом высокоинтерактивным и вынужденным образом. Для успеха на появляющемся глобальном рынке пожизненного обучения университету нужно создать портал кампуса, который достигнет такой степени интерактивности, пользовательской дружественности и персонализации, которых сегодня нет у громадного большинства Web-сайтов кампусов.

2. АДАПТИВНЫЕ ГИПЕРМЕДИА-СИСТЕМЫ

Адаптивные гипермедиа-системы повышают функциональность гипермедиа-систем. Целью этих систем является *персонализация* гипермедиа-систем, ее настройка на особенности индивидуальных пользователей. Таким образом, каждый пользователь имеет свою собственную картину и индивидуальные навигационные возможности для работы с гипермедиа-системой.

Адаптивные гипермедиа-системы сводят воедино идеи гипермедиа-систем и интеллектуальных обучающих систем. Они принадлежат к группе адаптируемых пользователем систем, таких как адаптируемый пользовательский интерфейс или интерфейс, основанный на модели пользователя. Более подробное обсуждение адаптивных гипермедиа-систем в контексте адаптируемых пользователем систем можно найти в [57].

Адаптивные системы используют модель пользователя для сбора информации о его знаниях, целях, опыте и т.д. для адаптации содержания и навигационной структуры. Приведем пример. Для пользователя с невысоким уровнем знаний может быть полезно вначале прочитать более общую вводную информацию, прежде чем углубляться в детали. Однако эта же информация не будет столь же интересной для эксперта. Здесь выбор нужной информации в нужное время является задачей для модели пользователя. Другой пример: информационная система по туризму должна рассматривать способности и ограничения своих пользователей. Если пользователь, имеющий физические недостатки, запрашивает время начала работы муниципалитета, ответ системы должен содержать также сведения о ближайших парковках или остановках общественного транспорта, также как и о возможности въезда в здание на коляске и т.п. (примеры см. в [44, 66, 72, 84]).

Адаптация гипермедиа-систем является также попыткой преодолеть возможность «заблудиться в гиперпространстве» (см. обсуждение этой проблемы в [68]). Цели и знания пользователя могут быть использованы для ограничения количества возможных ссылок гипермедиа-системы.

По возможностям адаптации различаются следующие типы гипермедиа-систем [30]:

1. *Адаптированные (приспособленные) гипермедиа-системы* (adapted hypermedia systems) — системы, в которых адаптация привносится в систему самим разработчиком после фазы тестирования. В этом случае адаптация не может быть корректной для каждого отдельного пользователя.

2. *Адаптируемые (приспосабливаемые) гипермедиа-системы* (adaptable hypermedia systems) — системы, которые могут модифицироваться только по явному требованию пользователя. Адаптируемые системы позволяют пользователю явно устанавливать предпочтения или предоставляют профиль через заполнение формы. Вся информация, предоставленная пользователем, хранится в модели пользователя, которая модифицируется только по явному запросу пользователя. Представление информации затем адаптируется к этой модели. Некоторые системы могут иметь очень сложные модели пользователя, в то время как другие различают только несколько стереотипных пользователей типа «начинающего», «среднего» и «эксперта».

3. *Адаптивные гипермедиа-системы* (adaptive hypermedia systems) — системы, которые сами могут адаптироваться к потребностям пользователя. Адаптивные системы формируют модель пользователя, отслеживая навигацию пользователя в информационном пространстве, а также посредством тестов в системах обучения. Представление адаптируется к модели пользователя, и модель пользователя постоянно обновляется, по мере того как пользователь просматривает информацию.

Таким образом, класс адаптивных гипермедиа-систем состоит из всех таких гипертекстовых и гипермедиа-систем, которые отражают некоторые особенности пользователя в модели пользователя и применяют эту модель для адаптации различных видимых для пользователя аспектов системы.

Поддержка адаптивных методов в гипермедиа-системах оказывается весьма полезной в тех случаях, когда имеется *одна* система, обслуживающая *множество* пользователей с *различными* целями, уровнем знаний и опытом, и когда лежащее в ее основе гиперпространство является относительно *большим* [17].

Обучающие гипермедиа-системы, в которых пользователь или ученик имеет конкретную цель обучения (включая и такую цель, как общее образование), являются типичным приложением адаптивных гипермедиа-систем. В этих системах основное внимание уделяется знаниям обучающихся, которые могут сильно различаться. Состояние знаний изменяется во время работы с системой. Таким образом, корректное моделирование изменяющегося состояния знаний, надлежащее обновление модели и способность делать правильные заключения на базе обновленной оценки знаний являются важнейшей составляющей обучающей гипермедиа-системы.

Отметим, что области применения выходят далеко за границы обучающих систем. Например, другим важным приложением являются *онлайновые информационные системы* (on-line information systems), а также *онлайновые справочные системы* (on-line help systems). К онлайновым информационным системам относятся, например, электронные энциклопедии, хранилища документов или туристические справочники. Чтобы выдать правильную информацию пользователям с различным уровнем квалификации, этим системам также требуется модель знаний пользователя. Важен также контекст запроса: нужна ли информация пользователю для краткой справки, для разработки презентации, для освежения знаний? Онлайновые справочные системы принимают во внимание конкретную среду, например, место вызова (контекстно-зависимые справочные системы).

Для ограничения вариантов навигации адаптивные гипермедиа-системы могут быть объединены со средствами поиска информации [59] в *гиперме-*

диа-системы с поиском информации. Ссылки в таких системах не вводятся автором системы, а формируются на основе сходства: ссылка между двумя документами устанавливается в том случае, если оба документа удовлетворяют некоторому условию похожести.

Подробное рассмотрение различных областей приложения адаптивных гипермедиа-систем сделано в работе [13] (см. также [1, 2]).

3. К ЧЕМУ АДАПТИРУЮТСЯ АДАПТИВНЫЕ ОБУЧАЮЩИЕ СИСТЕМЫ?

Адаптивная гипермедиа-система собирает информацию о пользователях. На основе их индивидуальных характеристик она адаптирует свое содержание и навигационные возможности к конкретному пользователю (рис. 1).

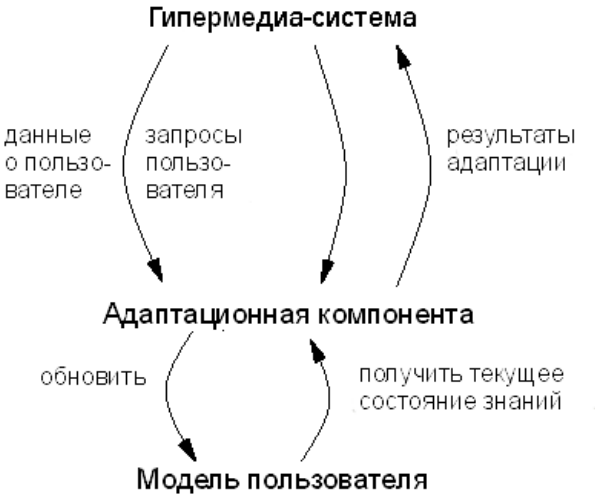


Рис. 1. Схематическое представление адаптивной гипермедиа-системы

Первый вопрос, возникающий при рассмотрении модели конкретной адаптивной системы, состоит в следующем: какие характеристики пользователя могут быть приняты во внимание для обеспечения адаптации? К каким параметрам (различным для различных пользователей и, возможно,

различным для одного и того же пользователя в разные моменты времени) система может адаптировать свое поведение? Вообще, существует довольно много параметров, связанных как с конкретным видом текущей деятельности пользователя, так и с пользователем как таковым, которые адаптивная система может принять во внимание: знания пользователя, его цели, предпочтения, подготовка, опыт и скорость обучения. Некоторые характеристики среды, в рамках которой пользователь осуществляет связь с Web-системой, могут также использоваться системой для адаптации, хотя и не относятся к самим пользователям, как таковым, и не могут быть выведены из характеристик пользователя.

Некоторые авторы [3] помимо адаптации к отдельному пользователю рассматривают еще адаптации к множествам пользователей, выделяя при компьютерном обучении три иерархических уровня адаптации к обучаемым.

1. Адаптация к студентам как категории пользователей.
2. Адаптация к группе студентов.
3. Адаптация к отдельному студенту.

Первый уровень адаптации предусматривает адаптацию к каждой категории пользователей компьютерной системы обучения в зависимости от их потребностей и обычно реализуется созданием специального интерфейса для каждого класса пользователей. Такой подход характерен для любых компьютерных систем. В интеллектуальных обучающих системах учащемуся необходимо предоставить следующие возможности: обучение, проверка знаний, упражнения, помощь и справочная информация, видеолекции и их презентации, вопросы преподавателю, конференции, студенческие форумы, электронные методические пособия, ввод комментариев по ходу занятия и др.

Адаптация к группе студентов предполагает адаптацию в зависимости от выбранной специальности, образовательной программы, возраста и психологической направленности личности. Этот уровень адаптации базируется, в первую очередь, на решении двух основных вопросов дидактики: «чему учить?» и «как учить?». Ответ на первый вопрос определяет цели обучения, т.е. объем необходимых знаний, умений и навыков и степень их освоения. Решение второго вопроса дидактики («как учить?») обуславливает выбор методов обучения, наиболее подходящих для группы учащихся, а также способов представления информации. На выбор методов обучения и способов представления информации влияют как возраст обучаемого, так и его психологическая направленность личности (ориентация на себя, на задачу, на взаимодействие).

3.1. Цели пользователя

Цель пользователя (или *задача пользователя*) — это параметр, зависящий в большей степени от самой природы текущей работы пользователя в данной гипермедиа-системе, нежели от пользователя как такового. В зависимости от типа системы, это может быть рабочая цель (в прикладных системах), цель поиска (в информационно-поисковых системах) или цель обучения или решения (в обучающих системах). Во всех этих случаях цель является ответом на вопрос: «Почему пользователь использует гипермедиа-систему и чего пользователь хочет в итоге достичь?». Цель пользователя — это наиболее изменчивая его характеристика. Она почти всегда будет другой при новом сеансе работы с системой, а иногда может изменяться и во время одного сеанса работы.

В обучающих системах целесообразно различать *локальные*, или цели *нижнего уровня*, которые могут изменяться достаточно часто, и *общие*, или цели (задачи) *высокого уровня*, являющиеся более стабильными. Например, цель изучения курса или некоторой темы курса — это цель высокого уровня, которая сможет сохраняться в течение нескольких сеансов, в то время как цель решения задачи — цель нижнего уровня, которая может изменяться от одной учебной задачи к другой несколько раз за один сеанс работы.

Цель пользователя рассматривается как очень важная его характеристика в адаптивных гипермедиа-системах. Почти третья часть существующих технических приемов адаптации основывается на этой характеристике. Интересно, что почти все эти приемы относятся к технологиям адаптивной поддержки навигации.

3.2. Знания пользователя

В существующих в настоящее время адаптивных гипермедиа-системах уровень знаний пользователя является наиболее важной информацией для адаптации. Особенно учет уровня знаний важен в образовательных системах, где изменение состояния знаний пользователя является важнейшей составляющей адаптации. Система должна всегда обновлять свою оценку уровня знаний пользователя, и адаптационная компонента должна использовать реальное состояние знаний для выполнения шагов адаптации.

Эта характеристика используется при реализации примерно одной трети технических приемов адаптации. Почти все приемы адаптивного представления используют знания пользователя как источник адаптации. Уровень знаний является переменной величиной для каждого конкретного пользователя. Это означает, что адаптивная гипермедиа-система, использующая зна-

ния пользователя, должна фиксировать изменения уровня этих знаний и соответствующим образом модифицировать модель пользователя.

3.3. Уровень подготовки пользователя и имеющийся у него опыт

Уровень подготовки пользователя и имеющийся опыт работы пользователя с данной гипермедиа-системой — это две характеристики пользователя, которые имеют нечто общее с уровнем знаний пользователя, но функционально отличаются от него.

Под *уровнем подготовки пользователя* понимается вся та информация, связанная с предыдущим опытом работы пользователя, которая не относится к теме данной гипермедиа-системы. Подготовка включает в себя профессию пользователя, опыт работы в смежных областях, а также точку зрения пользователя и его перспективу.

Опытом работы пользователя в данной гипермедиа называется характеристика, отражающая то, насколько хорошо пользователь знаком со структурой гипермедиа и то, насколько легко он может осуществлять навигацию в данной гипермедиа. Это не то же самое, что уровень знаний пользователя по теме, представленной в гипермедиа [89]. Иногда бывает так, что пользователь хорошо знаком с темой, но абсолютно не знаком со структурой гиперпространства. И наоборот, пользователь может быть хорошо знаком со структурой гиперпространства, но не иметь глубоких знаний по теме, представленной в нем. Еще одной причиной того, что необходимо различать опыт работы в гипермедиа и уровень знаний пользователя, является существование технического приема адаптивной поддержки навигации [78, 89], который основан на этой характеристике пользователя.

Такие индивидуальные характеристики пользователя, как уровень подготовки или опыт, обычно моделируются стереотипной моделью пользователя (MetaDoc, Anatom-Tutor, EPIAIM, C-Book). Стереотип может быть стереотипом опыта [78], [89] или стереотипом уровня подготовки для таких категорий, как профессия [39], перспектива [9] или родной язык.

3.4. Предпочтения пользователя

Важной характеристикой пользователя, рассматриваемой системами адаптивной гипермедиа, является *набор (система) предпочтений пользователя*. По различным причинам пользователь может предпочитать одни узлы, ссылки и части страниц гипермедиа другим. Эти предпочтения могут быть абсолютными (Hypadapter, Information Islands) или относительными, т. е. зависящими от текущего узла, цели (PUSH, HYPERFLEX) или текуще-

го контекста вообще (WebWatcher, CID, DHS, Adaptive HyperMan). Предпочтения наиболее тяжело использовать в поисковой гипермедиа. В то же самое время, в наиболее адаптивных поисковых гипермедиа-системах предпочтения — единственная накапливаемая информация о пользователе.

Некоторые предпочтения, например, по отношению к шрифту, иллюстрациям, примерам не могут быть определены системой без ввода данных пользователем. Системы, которые отражают различные типы предпочтений, позволяют своим пользователям настраивать эти свойства. При этом предполагается, что пользователь не склонен часто менять свои предпочтения.

3.5. Личностные характеристики пользователя

Данная группа характеристик связана с определением пользователя как индивидуума. Примерами личностных характеристик являются *персональные факторы* (например, интроверт или экстраверт), *когнитивные факторы* (например, *скорость обучения*) и *стили обучения*.

Подобно характеристикам пользовательской подготовки личностные характеристики пользователя являются его стабильными чертами, которые либо совсем не могут быть изменены, либо для своего изменения требуют длительный период времени. Однако в отличие от характеристик подготовки пользователя личностные характеристики как правило извлекаются не в процессе простого опроса, а с помощью специально разработанных психологических тестов. Многие исследователи согласны с важностью использования для адаптации личностных характеристик пользователя, но пока среди них мало согласия в том, какие характеристики можно и нужно использовать или как их использовать. В качестве примера можно привести ряд систем, в которых сделана попытка адаптации к стилю обучения [25, 29, 45, 87], отметив, что еще нет достаточной ясности в том, какие аспекты стиля обучения полезно моделировать и что можно сделать по-разному для пользователей с разными стилями.

3.6. Адаптация к среде пользователя

Поскольку пользователи одного и того же серверного Web-приложения могут находиться виртуально повсеместно и могут использовать различное оборудование, важной задачей становится адаптация к пользовательской среде. Имеется целый ряд систем, предлагающих некоторые технологии для адаптации как к местонахождению пользователя, так и к его платформе. Простая адаптация к платформе (оборудование, программное обеспечение, пропускная способность сети) обычно включает выбор типа представления

содержимого: материал или медиа (т.е. неподвижная картинка против ролика) [58]. Более продвинутые технологии могут обеспечить существенно более разный интерфейс для пользователей с разными платформами и даже использовать платформные ограничения для получения преимуществ от моделирования пользователя. Например, одна из версий системы AIS требует, чтобы пользователь явно запрашивал следующие страницы истории в новостях, посылая сообщения системе о том, какая история представляет интерес.

4. ЧТО АДАПТИРУЕТСЯ В СИСТЕМАХ WEB-ОБУЧЕНИЯ?

Сетевые обучающие системы успешно объединяют технологии адаптации, используемые в интеллектуальных обучающих системах и адаптивных гипермедиа-системах.

Целью различных интеллектуальных обучающих систем является использование знаний о сфере обучения, обучаемом и о стратегиях обучения для обеспечения гибкого индивидуализированного изучения и обучения. Для достижения этого ими традиционно используются следующие основные технологии (см., например, в [17]): построение последовательности курса обучения, интеллектуальный анализ ответов обучаемого и интерактивная поддержка в решении задач. Все эти три технологии можно рассматривать технологиями интеллектуальной адаптации сетевых обучающих систем. К группе технологий интеллектуальных адаптаций сетевых обучающих систем следует отнести также технологию, получившую название подбора моделей обучаемых (или просто подбором моделей) [11]. Суть ее состоит в анализе и подборе модели для многих обучаемых одновременно, и поэтому она почти не имеют корней в несетевых (доинтернетовских) образовательных системах, поскольку обычные адаптивные и интеллектуальные образовательные системы работают с одним обучаемым (и одной моделью обучаемого) за раз.

Что касается гипермедиа-систем, то в них область адаптации весьма ограничена и существует не так уж много параметров, которые можно изменять. С общей точки зрения гипермедиа-система состоит из набора узлов или *гипердокументов* (для краткости, будем называть их "страницами"), связанных ссылками. Каждая страница содержит некоторую локальную информацию и несколько ссылок на релевантные страницы. Системы гипермедиа могут также содержать индексную структуру и глобальную карту, которые обеспечивают доступ по ссылкам ко всем возможным страницам.

Поэтому адаптация в адаптивной гипермедиа может состоять в настройке содержания очередной страницы (адаптация на уровне содержания) или в изменении ссылок с очередной страницы, индексных страниц и страниц карт (адаптация на уровне ссылок). Поэтому различаются адаптации на уровне содержания и на уровне ссылок как два различных класса гипермедиа-адаптации, первый из которых называется *адаптивным представлением* (adaptive presentation), а второй — *адаптивной поддержкой навигации* (adaptive navigation support).

4.1. Выстраивание программы обучения

Цель технологии *выстраивания последовательности обучения* (curriculum sequencing), также называемой *технологией учебного планирования* (instructional planning technology), — помочь обучаемому найти свой «оптимальный путь» через учебный материал. Для этого нужно обеспечить обучаемого наиболее подходящей, индивидуально спланированной последовательностью блоков знаний для изучения и последовательностью учебных заданий (примеров, вопросов, задач и т.д.) для их выполнения.

Различают два вида выстраивания: активный и пассивный. *Активное* выстраивание подразумевает наличие *цели обучения* (подмножество понятий или тем, которыми надо овладеть). Системы с активной последовательностью могут построить лучший индивидуальный путь для достижения цели. *Пассивное* выстраивание (которое также называется *корректировкой*) является реактивационной технологией и не требует активной цели обучения. Она начинает действовать, когда пользователь не способен решить задачу или ответить правильно на вопрос (вопросы). Ее цель — предложить пользователю подмножество доступного материала для изучения, которое может заполнить пробел в знаниях студента для разрешения непонимания. В системах с активной последовательностью различают системы с жесткой и приспосабливаемой целью обучения. Большинство существующих систем могут провести своих студентов к фиксированной цели обучения — полному множеству понятий сферы обучения. Несколько систем с приспосабливаемой целью позволяют учителю или студенту выбирать подмножество всех понятий сферы обучения как текущую цель.

Разделяют два уровня выстраивания: высокий и низкий. Выстраивание *высокого уровня* (или *выстраивание знаний*) определяет следующую подцель для изучения: следующее понятие, набор понятий, тему или урок, которые необходимо выучить на следующем шаге. Выстраивание *низкого уровня* (или *выстраивание заданий*) определяет следующее учебное задание

(задачу, пример, тест) внутри текущей подцели. Последовательности высокого и низкого уровней часто исполняются различными механизмами. Во многих системах обучения только один из этих двух механизмов интеллектуальный, например, урок выбирает обучаемый, в то время как учебные задания внутри этого урока выбирает система. Некоторые системы могут управлять только порядком заданий конкретного вида: обычно задач или вопросов.

Активные выстраивания преобладают в существующих учебных системах. Только несколько систем (InterBook, PAT-InterBook, CALAT, VC Prolog Tutor, and Remedial Multimedia System) могут осуществлять пассивные корректирующие выстраивания. Среди систем с активными выстраиваниями только единицы, такие как ELM-ART-II — и KBS-Hyperbook, способны осуществлять выстраивания как высокого, так и низкого уровней. Большинство систем обучения поддерживает выстраивания с фиксированными целями (равнозначными полному курсу). Только несколько систем поддерживают приспосабливаемые цели изучения, позволяющие преподавателю (как в DCG) или обучаемому (как в InterBook и KBS Hyperbook) выбирать индивидуальную цель. Обучаемый может выбрать цель в виде подмножества понятий области обучения (InterBook) или проекта (KBS Hyperbook).

Активная последовательность в большинстве систем управляется знаниями обучаемого (точнее разницей между знаниями обучаемого и глобальной целью). Однако несколько систем и проектов экспериментируют с использованием предпочтений обучаемого в типе и способах представления доступного учебного материала для управления последовательностью заданий внутри темы [25, 29].

4.2. Интеллектуальный анализ решений обучаемого

Технология *интеллектуального анализа решений обучаемого* имеет дело с окончательными ответами студента на обучающие задания (которые могут колебаться от простых вопросов до сложных задач программирования) без разъяснения причин, по которым ответ был получен. В отличие от неинтеллектуальных проверяющих программ, которые не могут сказать ничего кроме того, что правильный ответ или нет, интеллектуальные анализаторы могут подсказать обучаемому, что именно неправильно или неполно и какие отсутствующие или неверные знания ответственны за ошибку.

Интеллектуальные анализаторы могут предоставлять обучаемому интенсивную обратную связь об ошибках и корректировать модель обучаемого.

Классический пример из области обучения программированию — это система PROUST. Интеллектуальный анализ решений — очень подходящая технология для медленных сетей. В рамках этой технологии необходимо только одно взаимодействие между браузером и сервером для принятия окончательного решения. Она может предоставлять интеллектуальную обратную связь и выполнять моделирование студента, когда другие интерактивные техники использовать затруднительно. В качестве примера укажем на две адаптивные обучающие Web-системы, которые реализуют интеллектуальный анализ решений обучаемого адаптивно в WWW (т.е. обучаемые с различными моделями могут получать различную обратную связь): интеллектуальную систему ELM-ART для обучения программированию на языке LISP [22] и интеллектуальную системы WITS для обучения дифференциальному исчислению [75].

4.3. Поддержка в решении задач

В течение многих лет поддержка в решении задач рассматривалась главной обязанностью интеллектуальных обучающих систем и их основным достоинством. Интеллектуальные обучающие системы используют три технологии поддержки в решении задач: интеллектуальный анализ решений обучаемого, интерактивная поддержка в решении задач и поддержка в решении задач на примерах. Все эти технологии позволяют помочь студенту в процессе решения учебной задачи, но делают они это разными способами.

Интеллектуальный анализ решений обучаемого имеет дело с конечными ответами обучаемого на учебные задачи (как были получены эти ответы неважно). Чтобы считаться интеллектуальным, анализатор решений должен не только уметь оценить правильность решения, но и найти, что в решении конкретно неправильно или неполно, и, возможно, определить, какие недостающие или неправильные знания могут быть ответственны за ошибку (последнее действие относится к определению знаний). Интеллектуальные анализаторы могут предоставлять обучаемым далеко идущую обратную связь и обновлять модель обучаемого. Классическим примером считается система PROUST.

Интерактивная поддержка в решении задач более современная и более мощная технология. Вместо ожидания конечного решения эта технология предоставляет обучаемому интеллектуальную помощь на каждом шаге решения задачи. Уровень помощи может быть разным: от оповещения о неправильно сделанном шаге до выдачи совета и выполнения следующего шага за студента. Системы (часто называемые *интерактивными тренаже-*

рами), в которых реализуется эта технология, могут наблюдать за действиями студента, понимать их и использовать их понимание для предоставления помощи и обновления модели обучаемого. Классический пример — это система LISP-TUTOR.

Технология *поддержки в решении задач на примерах* — самая новая. Эта технология помогает обучаемым решать новые задачи, не выделяя их ошибки, а предлагая примеры успешного решения схожих задач из их более раннего опыта (это могут быть примеры, объясненные им, или задачи, решенные ими ранее). Пример — это система ELM-PE.

В области традиционных интеллектуальных обучающих систем технология поддержки в решении задач на примерах абсолютно преобладает. Интерактивная поддержка в решении задач является главной целью почти всех интеллектуальных обучающих систем, в то время как интеллектуальный анализ решений обучаемого обычно, если и используется, то весьма несовершенен, а поддержка в решении задач на примерах вообще используется крайне редко. Но для сетевых адаптивных интеллектуальных систем ситуация меняется: и интеллектуальный анализ решений обучаемого, и поддержка в решении задач на примерах кажутся для них очень естественными и полезными. Обе технологии пассивны (работают по запросу обучаемого) и соответственно могут быть относительно легко реализованы в сети при помощи интерфейса CGI. Более того, старые однопользовательские программы адаптивных интеллектуальных обучающих систем, использующие эти технологии, могут быть относительно легко размещены в сети реализацией CGI-шлюзов к ним. Неудивительно, что эти технологии были среди первых, реализованных для сетевых систем. Важным преимуществом применения этих двух технологий в сети является их низкая интерактивность: им обеим обычно требуется только одно взаимодействие между браузером и сервером для цикла решения задачи. А это очень важно в случае медленной Интернет-связи. Эти технологии могут обеспечивать интеллектуальную поддержку в случае, когда трудно использовать более интерактивные технологии. В настоящее время эти технологии преобладают в сетевой среде над более мощной, но жадной до взаимодействия интерактивной поддержкой в решении задач.

Технология интерактивной поддержки в решении задач последней из технологий интеллектуальных обучающих систем переселилась в сеть. Проблема в том, что подход к реализации сетевых обучающих систем «наспех» (разработка интерфейсов CGI к старым однопользовательским интеллектуальным обучающим системам), использовавшийся в первых системах, не проявил себя должным образом для этой технологии. Это можно проде-

монстрировать на примере системы PAT-Online [82], которая вероятно была первой попыткой реализации интерактивной поддержки в решении задач в сети. Эта система использует основанный на разновидности CGI-AppleScript интерфейс для однопользовательской системы Репетитор по практической алгебре (Practical Algebra Tutor — PAT). Поскольку интерфейс CGI пассивный, сетевая версия должна предоставлять обучаемому кнопку «подчинения» для получения обратной связи с системой. Естественно это также добавляло еще одну особенность, которая была важна для обучаемых с медленным Интернет-соединением: возможность требования обратной связи после выполнения нескольких шагов решения задачи. В результате PAT-Online переместилась в категорию интеллектуальных анализаторов задач, точнее в подкатегорию анализаторов, способных анализировать неполные решения (ELM-ART также принадлежит к этой подкатегории). Интеллектуальные анализаторы этой подкатегории можно расположить между традиционными анализаторами и интерактивными репетиторами.

Настоящий интерактивный репетитор должен быть не только интерактивным, но и активным. Он не должен спать в промежутках от одного запроса к другому, а вместо этого он должен быть способен наблюдать, что делает обучаемый и немедленно реагировать на ошибки. Но это просто не может быть реализовано при помощи обычной интерактивности CGI на сервере и требует клиентской интерактивности, основанной на Java. Java обеспечивает надежное решение проблемы для сетевых интерактивных репетиторов. Если быть более точным, то Java предлагает два различных решения. Одно состоит в том, что репетитор полностью реализован на Java. Это может быть как апплет, работающий в браузере, так и приложение Java. Другим решением является распространяемый клиент-серверный репетитор, в котором часть функций реализована на Java и работает на клиентской стороне, а другая часть работает на сервере. Части связаны через Интернет. Хотя чистое Java-решение выглядит проще (всего лишь новый язык для создания адаптивных интеллектуальных обучающих систем), клиент-серверная архитектура предлагает более привлекательный выбор для развития сетевых репетиторов. Это определенный выбор для размещения однопользовательских интерактивных репетиторов в Сети. D3-WWW-Trainer [42] и AlgeBrain [7] показывают, как заново использовать интеллектуальные функции предшествующих однопользовательских репетиторов, заменяя их на серверные приложения и разрабатывая относительно слабых "безмозглых" Java-клиентов, которые реализуют интерфейсные функции и связываются с интеллектуальным сервером. Такие событийные репетиторы, как

ADIS [92] и ILESA [67], которые могут быть легко реализованы на чистой Java, в отличие от клиент-серверной архитектуры имеют такое преимущество как централизованное построение модели обучаемого. В конечном итоге накладные расходы клиент-серверного подхода (необходимость иметь распространяемую систему) не такие большие, с тех пор как Java естественным образом поддерживает несколько способов клиент-серверной взаимосвязи — HTTP/CGI, сокеты или RMI/CORBA. Считается, что клиент-серверная архитектура становится стандартом реализации сетевых интерактивных репетиторов и весьма популярным средством для реализации всех видов высоко интерактивных сетевых адаптивных интеллектуальных обучающих систем. Мы уже видим примеры ее использования: в реализации интерфейса, основанного на пере в WITS-II [55], и одушевленного педагогического агента Винсента в TEMAI [76].

4.4. Подбор моделей обучаемых

Суть технологии *подбора моделей обучаемых* состоит в способности анализировать и подбирать модели для многих обучаемых одновременно. В обычных (несетевых) интеллектуальных образовательных системах она не работает, поскольку несетевая система обычно работает с одним обучаемым (и одной моделью обучаемого) при каждом ее применении. Иначе обстоит дело в случае сетевых систем, для которых возможность подбора моделей естественна, поскольку записи обучаемых хранятся централизованно на сервере (по крайней мере, в административных целях). Это обеспечивает отличную основу для развития различных адаптивных и интеллектуальных технологий, которые смогут как-то использовать подбор моделей для различных обучаемых.

Используются два вида подбора моделей обучаемых: адаптивная поддержка сотрудничества и интеллектуальное наблюдение за классом. Эти виды полностью отличаются друг от друга и рассматриваются как различные технологии внутри группы подбора моделей обучаемых [11]. Целью *адаптивной поддержки сотрудничества* является использование знаний системы о разных обучаемых для подбора групп для различных видов сотрудничества (см., например, [48], [54]). *Интеллектуальное наблюдение за классом* также основано на возможности сравнивать записи о разных обучаемых, но вместо поиска совпадений ищутся различия. Цель сравнения — определение тех обучаемых, которые учатся существенно отличающимся образом от своих сокурсников. Эти обучаемые могут отличаться от остальных по-разному. Они, например, могут развиваться слишком быстро или

слишком медленно или просто осуществить доступ к гораздо меньшему материалу, чем остальные. В любом случае эти обучаемые нуждаются в большем внимании преподавателя, чем остальные, для того чтобы бросить вызов тем, кто может, дать больше объяснений тем, кто не может, а также подтолкнуть тех, кто мешкает. В обычной аудитории преподаватель может следить за посещаемостью и вниманием обучаемых, чтобы найти обучаемых, нуждающихся в особом внимании. В сетевой аудитории преподаватель в лучшем случае имеет только данные из журнала, из которых сложно сделать правильные выводы. В то же время необходимость распознавания небольшого подмножества обучаемых, нуждающихся в помощи больше, чем остальные, является весьма важной. В сетевой среде обучения на общение между преподавателем и обучаемыми обычно тратится больше времени, чем в обычной аудитории, и отдаленный преподаватель может индивидуально обратиться лишь к небольшому числу обучаемых. Система HyperClassroom представляет интересный пример использования нечетких механизмов для определения застоявшихся обучаемых в аудитории сетевой обучающей системы.

4.5. Адаптация на уровне содержания, цели адаптации

Основная идея всех технологий адаптации на уровне содержания состоит в том, чтобы адаптировать представление страницы, к которой обращается пользователь, к его текущему уровню знаний, целям и другим характеристикам.

Адаптивное представление текста позволяет изменить текстовое содержание гипермедиа-страницы в зависимости от модели пользователя. Например, за счет адаптации текста квалифицированный пользователь может быть обеспечен более детальной и полной информацией, в то время как новичок может получать дополнительные объяснения.

В гипермедиа-системах страница может содержать не только текст, как в классических гипертекстовых системах, но и всевозможную мультимедиа-информацию. *Адаптация мультимедиа* предполагает, что элементы мультимедиа-содержания могут быть адаптированы к индивидуальному пользователю, но текущие реализации ограничены выбором средств; в отличие от текста, содержание анимации, аудио- или видеофрагментов трудно адаптировать.

А*даптация модальности* — новая технология адаптации содержания высокого уровня [13]. Современные адаптивные гипермедиа-системы могут иметь выбор различных типов средств для представления информации

пользователю, т. е. в дополнение к традиционному тексту может использоваться музыка, видео, речь, анимация и т.д. Весьма часто фрагменты различных средств представления информации имеют то же самое содержание, и, следовательно, система может выбрать тот, который наиболее релевантен пользователю для данного узла. В других случаях эти фрагменты могут использоваться параллельно, таким образом позволяя системе выбрать наиболее релевантное подмножество элементов средств представления информации. В настоящее время можно идентифицировать несколько различных методов для адаптации модальности представления на основе предпочтений пользователя, способностей, стиля изучения и контекста работы в нескольких видах адаптивных гипермедиа-систем.

В процесс адаптации на уровне содержания происходит «подгонка» содержания документа под конкретного пользователя, к примеру, за счет скрытия слишком специализированной информации или за счет вставки дополнительных объяснений.

Согласно [17] можно выделить следующие цели (методы) адаптации на уровне содержания.

1. *Дополнительные объяснения.* Только те части документа должны выдаваться пользователю, которые соответствуют его уровню знаний или его цели.

2. *Предварительные объяснения.* Цель — снабдить пользователя необходимой ему информацией для понимания документа. По модели пользователя проверяются предварительные знания, требуемые для понимания содержания страницы. Если пользователь не обладает какими-то знаниями, соответствующая информация должна быть добавлена к странице.

3. *Сравнительные объяснения*. Идея сравнительных объяснений состоит в пояснении новых тем путем подчеркивания их связей с уже известными темами.

4. *Варианты объяснений.* Предоставляя различные объяснения для некоторых частей документа, мы можем выбрать те из них, которые максимально подходят для данного пользователя. Этот подход является расширением метода предварительных объяснений.

5. *Сортировка.* Различные части документа сортируются в соответствии с их актуальностью для пользователя.

4.6. Технологии адаптации на уровне содержания

Адаптация на уровне содержания требует хорошо разработанных технологий для улучшения представления. Существующие системы, осуществ-

ляющие адаптацию на уровне содержания, достигают этого путем обогащения своих документов мета-информацией о предварительных условиях или требуемых знаниях, результатах и т.д. Документы или их фрагменты, хранящиеся в таких системах, должны включаться помногу раз, чтобы сделать возможными варианты объяснений.

Для достижения целей адаптации на уровне содержания используются следующие техники [17].

1. *Условный текст* (conditional text) предполагает, что вся информация о понятии разбивается на определенные текстовые куски. Для каждого из этих кусков определяется требуемый уровень знаний пользователя, при наличии которого производится его вывод.

Технология условного текста является технологией нижнего уровня, требующей от разработчика некоторой работы по «программированию» для установления всех требуемых условий. Выбирая соответствующие условия на уровне знаний текущего понятия и связанных с ним понятий, представленных в модели пользователя, можно реализовать все методы адаптации, перечисленные выше, за исключением сортировки. Простой пример — это скрытие частей текста с неподходящими объяснениями, если уровень знаний пользователя текущего понятия достаточно хорош, или включение части текста со сравнительными объяснениями, если соответствующее сходное понятие уже известно.

2. *Эластичный текст* (stretchtext). Технология более высокого уровня, которая может также включать и выключать различные части текста в соответствии с уровнем знания пользователей, когда некоторые ключевые слова документа могут заменять более длинные описания или заменяться ими, если этого требует реальный уровень знаний пользователя. В обычном гипертексте результатом активации ключевого слова является перемещение на другую страницу со связанным текстом. В эластичном тексте этот связанный текст может просто заменять активизированное слово (или фразу с этим словом), расширяя текст текущей страницы. Если требуется, этот расширенный или «развернутый» текст может быть свернут обратно в слово. Каждый узел — эластичная страница, которая может содержать много «свернутых» слов. Идея адаптивного эластичного представления состоит в том, чтобы представить требуемую страницу со всеми свернутыми расширениями, нерелевантными для пользователя, и всеми развернутыми расширениями, релевантными для пользователя.

Важная особенность данной адаптивной технологии в том, что она позволяет и пользователю, и системе адаптировать содержание отдельной страницы и что она может принимать во внимание и знания, и предпочте-

ния пользователя. После произвольного представления эластичной страницы, она может быть далее адаптирована пользователем, который может сворачивать или разворачивать соответствующие объяснения и подробности в соответствии со своими предпочтениями. Модель пользователя может обновляться в соответствии с демонстрируемыми пользователем предпочтениями, чтобы гарантировать, что пользователь всегда будет видеть предпочитаемую комбинацию сокращенных и несокращенных частей. Например, если пользователь свернул дополнительные объяснения отдельного понятия, они будут показываться свернутыми до тех пор, пока пользователь не изменит предпочтения.

3. *Варианты страниц или фрагментов страниц* — техники, позволяющие реализовывать метод вариантного объяснения.

Варианты фрагмента (fragment variants) — более мелкомодульная реализация метода *вариантного объяснения*. Система хранит несколько вариантов объяснения каждого понятия (варианты фрагментов), и пользователь получает страницу, содержащую те варианты, которые соответствуют его знанию о представленных понятиях. Технологии вариантов страницы и вариантов фрагмента могут быть скомбинированы, например, для обеспечения адаптации и к подготовке, и к знанию пользователя. Текущий вариант страницы для конкретного узла выбирается согласно подготовке пользователя. Эта страница затем может быть адаптирована: для каждого понятия, представленного на странице, система выбирает объяснение, которое наиболее соответствует уровню знаний пользователя.

Варианты страницы (page variants) — наиболее простая технология адаптивного представления. При использовании этой технологии система хранит несколько вариантов страницы с различными представлениями ее содержания. Как правило, каждый вариант подготовлен для одного из возможных стереотипов пользователя. При представлении страницы система выбирает вариант страницы согласно стереотипу пользователя.

4. *Технология, основанная на фреймах*, (frame-based technique) является наиболее мощной из всех технологий адаптации содержания. При использовании этой технологии вся информация об отдельном понятии представлена в форме фрейма. Слоты фрейма могут содержать несколько вариантов объяснения понятия, связи с другими фреймами, примерами и т.д. При использовании технологии естественного языка страницы монтируются из маленьких информационных элементов подобно словам и частям предложений. Используются специальные правила представления для решения того, какие слоты должны быть представлены конкретному пользователю и в каком порядке. Более точно, эти правила позволяют сделать выбор одной

из существующих схем представления (каждая схема — это упорядоченное подмножество слотов), которая и используется для представления понятия. Могут применяться правила для вычисления «приоритета представления» для каждого слота, и затем подмножество слотов с высоким приоритетом представляется в порядке уменьшения приоритета. В своих условных частях эти правила могут ссылаться не только на уровень знаний пользователем представляемого понятия, но также и на любую характеристику, представленную в модели пользователя. В частности, система может принимать во внимание подготовку пользователя.

Технология, основанная на фреймах, может использоваться для реализации всех методов, упомянутых выше. Методы предварительных и сравнительных объяснений могут быть реализованы с фреймовой технологией, при этом соответствующие условия ставятся на уровне знаний связанных понятий.

4.7. Адаптация на уровне ссылок: цели адаптации навигации

Используя адаптацию на уровне ссылок, можно персонализовать возможности навигации пользователя по гипермедиа-системе с учетом целей, знаний и других характеристик индивидуального пользователя. Техники адаптации на уровне ссылок приведены, к примеру, в [17]; они зависят от конкретной системы. В них предположения системы о пользователе играют важную роль для определения того, что и как адаптировать.

Адаптация на уровне ссылок ограничивает количество ссылок и, следовательно, объем навигационных возможностей. Она полезна для исключения ситуации «затерянности в гиперпространстве». Как и в случае адаптации на уровне содержания, для реализации задач адаптации требуется описание содержания документов.

Различают следующие пять целей адаптации навигации: глобальное руководство, локальное руководство, локальная ориентация, глобальная ориентация и управление индивидуализированными представлениями в информационных пространствах [16, 17]. Рассмотрим их более подробно.

1. *Глобальное руководство* (global guidance) можно обеспечить в гипермедиа-системах, в которых пользователи имеют некоторую «глобальную» информационную цель (т. е. нуждаются в информации, которая содержится в одном или нескольких узлах где-нибудь в гиперпространстве) и просмотр — способ нахождения требуемой информации. Цель методов глобального руководства состоит в том, чтобы помочь пользователю найти самый короткий путь к информационной цели с минимальным блужданием.

Глобальное руководство — основная цель адаптивной навигационной поддержки в информационно-поисковых гипермедиа-системах, а также важная цель в информационных и справочных системах с достаточно большим гиперпространством. Информационная цель пользователя, которая обычно полностью или частично предоставляется пользователем — основная характеристика пользователя для адаптивного руководства.

Наиболее полный метод обеспечения глобального руководства состоит в том, чтобы на каждом шаге просмотра предлагать пользователю ссылку для дальнейшего перехода (т. е. применять технологию *полного руководства*). Более поддерживающий метод состоит в применении технологии *адаптивной сортировки:* все ссылки от данного узла сортируются в соответствии с релевантностью глобальной цели (наиболее релевантные — сначала). При этом пользователи все еще имеют возможность перехода по первой наиболее релевантной ссылке, но также имеют некоторую информацию (релевантность других ссылок), чтобы сделать свободный выбор.

2. *Локальное руководство* (local guidance). Цель методов локального руководства состоит в том, чтобы помочь пользователю сделать один навигационный шаг, предлагая ссылки от текущего узла, наиболее подходящие для перехода. Эта цель отчасти подобна цели глобального руководства, но более «скромна». Методы локального руководства не предполагают глобальную цель для обеспечения руководства. Они делают указание согласно предпочтениям, знанию и подготовке пользователя — что наиболее важно для данной прикладной области. Например, подходящий метод локального руководства для ИП гипермедиа и сетевых информационных систем — это *сортировка* ссылок согласно предпочтениям и подготовке пользователя.

Методы, используемые в гипермедиа-системах обучения: сортировка ссылок согласно знанию пользователя и полное руководство в соответствии со знанием пользователя. Последний метод обычно применяется для выбора наиболее подходящей задачи из набора задач, доступных из текущего пункта.

3. *Поддержка локальной ориентации* (local orientation support). Цель методов поддержки локальной ориентации состоит в том, чтобы помочь пользователю в локальной ориентации (т. е. помочь пользователю понять, что находится вокруг и какова его относительная позиция в гиперпространстве). Существующие адаптивные гипермедиа-системы осуществляют поддержку локальной ориентации двумя различными способами: предоставляя дополнительную информацию об узлах, доступных от текущего узла (использование технологии *аннотирования*), и ограничивая число навигационных возможностей для уменьшения познавательной перегрузки, позволив

пользователям сфокусироваться на анализе наиболее релевантных ссылок (использование технологии *сокрытия*).

Методы, основанные на технологии *сокрытия*, имеют ту же самую идею: скрыть от пользователя все ссылки (или от индекса, или от локального узла), которые не подходят ему в данный момент, или, другими словами, показывать только релевантные ссылки (соответствующие знанию, цели, опыту или предпочтениям пользователя). Другой метод, основанный на *опыте* пользователя в данном гиперпространстве, состоит в том, чтобы показывать пользователям тем больше ссылок, чем больше опыта они имеют в гиперпространстве.

В гипермедиа-системах обучения применяются два специфичных метода, основанных на технологии *сокрытия:* смысл первого метода состоит в сокрытии ссылок к узлам, которые пользователь не готов еще изучить, а второго — к узлам, которые принадлежат целям обучения последующих уроков и не принадлежат текущей цели обучения.

Цель методов, основанных на технологии *аннотирования*, состоит в том, чтобы информировать пользователя о текущем «состоянии» узлов за видимыми ссылками. Аннотирование может использоваться, чтобы показать несколько градаций релевантности ссылки или несколько уровней знания пользователем узлов за аннотируемыми ссылками, а также чтобы выделить ссылки, соответствующие текущей цели, и затемнить несоответствующие.

Методы поддержки локальной ориентации не руководят пользователем непосредственно, но обеспечивают помощь в понимании того, что является ближайшими ссылками, а также в создании обоснованного навигационного выбора.

4. *Поддержка глобальной ориентации* (global orientation support). Цель методов поддержки глобальной ориентации состоит в том, чтобы помочь пользователю понять структуру всего гиперпространства и свою абсолютную позицию в нем. В неадаптивной гипермедиа эта цель обычно достигается, обеспечивая визуальные ориентиры и глобальные карты. Адаптивная гипермедиа может обеспечить большую поддержку для пользователя, применяя технологии *аннотирования* и *сокрытия*.

Аннотации функционируют как ориентиры: так как узел сохраняет ту же самую аннотацию, когда пользователь видит его из различных мест гиперпространства, пользователь легко может узнавать узлы, которые он встречал прежде, и понимать свою текущую позицию. *Сокрытие* уменьшает размер видимого гиперпространства и может упростить и ориентацию, и обучение, обеспечивая постепенное изучение гиперпространства. Перспек-

тивное направление адаптивной поддержки глобальной ориентации — *адаптация локальных и глобальных карт*, когда сама структура карты может зависеть от характеристик пользователя.

5. *Управление индивидуализированными представлениями* (managing personalized views). Индивидуализированные представления — способ организации электронного рабочего места для пользователей, которые нуждаются в доступе к достаточно небольшой части гиперпространства для своей повседневной работы. Каждое представление — просто список ссылок ко всем гипердокументам, которые соответствуют конкретной рабочей цели. Классические гипермедиа-системы и современные WWW-браузеры предлагают закладки и рабочие списки как способы делать персональные представления. Более развитые системы предлагают некоторые механизмы адаптируемости более высокого уровня, основанные на метафорах и моделях пользователя.

Адаптивные решения, т. е. управляемая системой поддержка индивидуализированных представлений, требуются в WWW-подобных динамических информационных пространствах, где объекты могут появляться, исчезать или изменяться. Для этого используются интеллектуальные агенты, которые могут регулярно искать новые релевантные объекты и распознавать измененные объекты или объекты с истекшим сроком хранения.

4.8. Технологии поддержки адаптивной навигации

Технологии адаптивной навигационной поддержки могут быть классифицированы согласно способу, который они используют для адаптации представления ссылок. Различаются шесть технологий для адаптирования представления ссылок: полное руководство (direct guidance), сортировка ссылок (link sorting), сокрытие ссылок (link hiding), аннотирование ссылок (link annotation), генерирование ссылок (link generation) и адаптация карты (map adaptation) [13, 15, 17, 30, 35, 36].

Для сравнения этих технологий сначала следует понять, как и в каком контексте обычно представляются ссылки. Здесь мы понимаем ссылки в смысле пользователя (т. е. видимое изображение связанных страниц, к которым пользователь может перейти). Определим четыре вида представления ссылок, которые отличаются тем, что в них может быть изменено и адаптировано [17]:

(1) *локальные неконтекстные (контекстно-независимые) ссылки.* Этот тип включает все виды ссылок на обычных гипермедиа-страницах, которые не зависят от содержания страницы. Они могут выглядеть как список, набор

кнопок или всплывающее меню. Этими ссылками просто управлять — они могут быть отсортированы, скрыты или аннотированы;

(2) *контекстные (контекстно-зависимые) ссылки или «настоящие гипертекстовые»* ссылки. Этот тип включает «горячие слова» (слова, связывающие текст с объектом) в текстах, «горячие точки» в изображениях и другие виды ссылок, которые встроены в контекст содержания страницы и не могут быть удалены из него. Эти ссылки могут быть аннотированы, но не могут быть отсортированы или полностью скрыты;

(3) *ссылки индексов и страниц содержания*. Индекс или страница содержания может рассматриваться как специальный вид страницы, который содержит только ссылки. Эти ссылки обычно представляются в фиксированном порядке (порядок содержания для страниц содержания и алфавитный порядок для страниц индексов). Как правило, ссылки с индексов и страниц содержания контекстно-независимые, если страница не реализована в форме изображения;

(4) *ссылки на локальных и глобальных картах гиперпространства*. Карты обычно графически представляют гиперпространство или локальную область гиперпространства как сеть узлов, связанных стрелками. При использовании карт пользователь может прямо перейти ко всем узлам, видимым на карте, просто «щелкая» по изображению желаемого узла. С навигационной точки зрения, эти изображения узлов — именно то, что мы понимаем под ссылками, в то время как стрелки, служащие изображением ссылок, не используются для прямой навигации.

Полное руководство, сортировка, сокрытие, аннотирование и адаптация карты, рассмотренные ниже, — основные технологии для адаптивной навигационной поддержки. Большинство из существующих технологий адаптации используют ровно один из этих способов для обеспечения адаптивной навигационной поддержки. Однако эти технологии не противоречивы и могут использоваться в комбинациях. В частности, технология полного руководства может легко использоваться в комбинации с любой другой технологией. Рассмотрим основные технологии поддержки адаптивной навигации более подробно.

1. *Полное руководство* (direct guidance) — наиболее простая технология адаптивной навигационной поддержки. Полное руководство может применяться в любой системе, которая может решить, какой следующий «наилучший» узел для посещения пользователем в соответствии с целью пользователя и другими параметрами, представленными в модели пользователя. Ссылка предоставляется к той странице, которую система считает наиболее подходящей для следующего перехода пользователя.

Различаются два метода полного руководства: метод *«следующего наилучшего»* (next best), который предполагает предоставление кнопки «next» для дальнейшей навигации через гиперпространство, и метод *установления последовательности страниц* или *следа* (page sequencing or trails), который предполагает генерирование последовательности страниц для просмотра всей гипермедиа-системы или ее части.

Полное руководство может использоваться со всеми четырьмя видами представления ссылок, перечисленных выше, но оно вряд ли может быть основной формой навигационной поддержки, поскольку не обеспечивает никакой помощи для пользователей, которые не хотели бы следовать указаниям системы.

2. *Адаптивная сортировка (упорядочение) ссылок* (adaptive sorting (ordering) of links) — технология, которая вместо одной «наилучшей» ссылки предоставляет список ссылок, упорядоченный по убыванию релевантности, в соответствии с моделью пользователя.

Различаются два метода сортировки: метод *сортировки по подобию* (similarity sorting), когда ссылки сортируются, исходя из их подобия данной странице, и метод *предварительных знаний* (prerequisite knowledge), когда ссылки упорядочиваются согласно предварительно необходимым знаниям.

Адаптивное упорядочение имеет ограниченную применимость: оно может использоваться только с контекстно-независимыми ссылками. Другая проблема с адаптивным упорядочением состоит в том, что эта технология делает порядок ссылок неустойчивым: он может изменяться каждый раз при входе пользователя на страницу. В то время как некоторые недавние исследования показали, что устойчивый порядок параметров в меню важен для начинающих. Тем не менее эта технология полезна для приложений информационного поиска.

3. *Адаптивное сокрытие ссылок* (adaptive link hiding) — в настоящее время наиболее часто используемая технология для адаптивной навигационной поддержки. Идея навигационной поддержки с помощью сокрытия состоит в том, чтобы ограничить навигационное пространство, скрывая ссылки к «нерелевантным» страницам. Страница может рассматриваться как нерелевантная по нескольким причинам: например, если она не связана с текущей целью пользователя или если она представляет материал, который пользователь не готов еще понять. Сокрытие защищает пользователей от запутанности неограниченного гиперпространства и уменьшает их познавательную перегрузку, также оно более понятно пользователю и выглядит более «стабильно» для них, чем адаптивное упорядочение.

Различают следующие три вида сокрытия ссылок: *сокрытие ссылок* (link hiding), *удаление ссылок* (link removal) и *отключение ссылок* (link disabling) [17], [30].

Адаптивное сокрытие ссылок делает анкер ссылки неотличимым от нормального текста. Сокрытие имеет широкую применимость: оно может использоваться со всеми видами контекстно-независимых, индексных ссылок и ссылками карт с помощью реального сокрытия кнопок или пунктов меню и с контекстно-зависимыми ссылками, превращая «горячие слова» в обычный текст.

Адаптивное удаление ссылок удаляет анкер ссылки для нерелевантных ссылок. Это может быть выполнено только тогда, когда окружающий ссылку текст таков, что он не становится бессмысленным после удаления ссылки. Хотя адаптивное удаление ссылок эффективно сокращает число навигационных шагов, предварительные оценки показывают, что пользователям не нравится эта технология.

Адаптивное отключение ссылок предполагает удаление функциональных возможностей ссылки без удаления текста ссылки. Отключение ссылки часто объединяется с сокрытием ссылки, потому что, если вид анкера остается, но ссылка не «работает», пользователь может подумать, что это ошибка в системе. Пользователям не нравится адаптивное отключение ссылок, но все же они предпочитают его адаптивному удалению ссылок.

4. *Адаптивное аннотирование ссылок* (adaptive link annotation) — технология, смысл которой состоит в пополнении ссылок некоторыми комментариями, чтобы дать пользователю подсказку относительно содержания страницы за аннотируемыми ссылками. Аннотации могут быть представлены в текстовой форме или в форме визуальных подсказок, например, используя различные значки, цвета или размеры шрифтов.

Наиболее популярный способ аннотирования ссылок — использование *метафоры светофора* (traffic light metaphor). Ссылка аннотируется цветной точкой: красная точка перед ссылкой указывает, что у пользователя недостаточно знаний для понимания этой страницы, таким образом, страница не рекомендуется для чтения. Желтая точка означает, что страница не рекомендуется для чтения, эта рекомендация менее строга, чем в случае красной точки. Зеленая точка ставится перед ссылками на страницы, рекомендуемыми для чтения. Серая точка указывает пользователю, что содержание этой страницы уже известно. Существуют и другие варианты.

Типичный вид аннотирования, рассматриваемый в традиционной гипермедиа — статическое (независимое от пользователя) аннотирование. Адаптивное аннотирование в самой простой, основанной на предыстории

форме (выделение ссылок к ранее посещенным узлам) применяется в некоторых гипермедиа-системах, включая большинство WWW-браузеров. Текущие адаптивные гипермедиа-системы могут различать и аннотировать поразному до шести состояний на основе модели пользователя.

Аннотирование может использоваться со всеми четырьмя возможными типами ссылок. Эта технология поддерживает стабильный порядок ссылок и избегает проблем с формированием неправильных ментальных карт. Аннотирование — более мощная технология, чем сокрытие: сокрытие может различать только два состояния узлов — уместные и неуместные, в то время как аннотирование — до шести состояний, в частности, несколько уровней релевантности. Эксперименты привели к заключению, что адаптивное аннотирование ссылок является полезным для сокращения числа навигационных шагов и в улучшении понимания учебного материала.

5. *Адаптивное генерирование ссылок* (adaptive link generation). Рост систем рекомендаций делает необходимым различать два по существу различных способа адаптивной навигационной поддержки: адаптация ссылок, представленных на странице во время разработки гиперпространства, и генерирование новых (неразработанных) ссылок для страницы. Генерирование ссылок включает три случая: обнаружение новых полезных ссылок между документами и добавление их к постоянному набору существующих ссылок, генерирование ссылок для навигации между элементами, основанной на подобии, и динамическая рекомендация релевантных ссылок.

Адаптивное генерирование ссылок было описано в [17] как новая технология адаптивной навигационной поддержки высокого уровня. Эта технология может использоваться вместе с другими технологиями типа аннотирования и сортировки.

6. *Адаптация* карты (map adaptation) — технология, которая включает различные пути адаптации формы глобальных и локальных гипермедиа карт (графические представления структуры ссылок), предоставляемых пользователю. Карты могут быть адаптивно фильтрованы, чтобы обеспечить представление частей гипердокумента, которые релевантны для пользователя. Такие технологии, как полное руководство, сокрытие и аннотирование, могут использоваться для адаптации гипермедиа-карты, но все эти технологии не изменяют форму или структуру карт.

5. АРХИТЕКТУРА АДАПТИВНОЙ ГИПЕРМЕДИА-СИСТЕМЫ

На абстрактном уровне в любой адаптивной гипермедиа-системе можно выделить следующие три основные взаимодействующие компоненты [37, 100].

- *Модель предметной области* (domain model), которая описывает, как структурировано информационное содержимое приложения (или гипердокумента). Эта модель должна указывать, какие имеются отношения между концептами и как концепты привязаны к информационным фрагментам и страницам.
- *Модель пользователя* (user model), которая представляет пользовательские предпочтения, знания, цели, историю навигации и другие относящиеся к пользователю характеристики. Система узнает дополнительную информацию о характеристиках пользователя, отслеживая его поведение. Одновременно система должна делать заключения о том, как изменяются соответствующие характеристики пользователя во время использования пользователем данного приложения.
- *Модель обучения* (teaching model), называемая также *моделью адаптации* (adaptation model), которая позволяет осуществить необходимые адаптации системы, например, с помощью так называемых *педагогических правил* (pedagogical rules) или *правил адаптации* (adaptation rules).

Следует отметить, что в большинстве существующих адаптивных систем указанные три составляющих не являются разделенными, что связано со следующими причинами [37, 100].

1. Отношения между страницами и концептами иногда слишком неопределены. Когда две страницы представляют 30% одного и того же концепта каждая, нет способа понять вместе они представляют 30%, 60% или любое другое значение между ними. В других системах это отношение строго взаимно однозначно, что приводит к большой фрагментации пользовательской модели без концепций высокого уровня.

2. Часто педагогические правила не могут быть определены на концептуальном уровне, поскольку они связаны со ссылками или (условными) текстовыми фрагментами, и поэтому должны специфицироваться в терминах реальных информационных страниц.

3. Имеется определенное несоответствие между высоким уровнем составляющих пользовательской модели и мало надежной информацией, на основании которой система должна изменять эту модель. Основная инфор-

мация, которая доступна большинству систем — это время, на которое затребовал пользователь некоторую страницу. Поэтому часто пользовательская модель является ненадежной только в силу того, что система получает ненадежные входные данные. Многие обучающие системы компенсируют данную ненадежность с помощью различных тестов.

Помимо трех моделей (предметной области, пользователя и адаптации) адаптивная гипермедиа система может содержать так называемый *механизм адаптации* (adaptive engine). Это — реальный программный продукт, являющийся частью системы, который используется ею для конструирования и адаптации содержимого и ссылок. Механизм поддерживает некоторую библиотеку функций для конструирования информационных страниц из фрагментов, основанных на элементах из моделей предметной области, пользователя и адаптации. Имеется язык (обычно довольно простой) для выбора «конструктора» для использования. Некоторые механизмы адаптации могут поддерживать способ определения новых конструкторов или расширения существующих. Однако, достаточно мощный механизм должен поддерживать достаточно стандартную функциональность для облегчения потребностей авторов явно специфицировать новые конструкторы в большинстве приложений. Механизм адаптации также изменяет пользовательскую модель, отслеживая поведение пользователя и, таким образом, принимая во внимание, как изменяются пользовательские знания.

5.1. Модели предметной области

Модель предметной области дает описание предметной области на концептуальном уровне и представляет собой совокупность объектов (концептов и межконцептных отношений), каждый из которых имеет уникальное имя.

Концепты — это абстрактные объекты, используемые для представления элементов информации предметной области. Существуют концепты *нижнего уровня* или *атомарные концепты*, которые соответствуют одному фрагменту информации, и концепты *высшего уровня* или *составные концепты*, которые состоят из множества других концептов [35, 37, 96, 99, 100].

Атомарные концепты являются первичными в модели и могут не подвергаться адаптации. Их атрибутные и анкерные значения принадлежат «внутрикомпонентному уровню» и таким образом являются зависимыми от реализации и не описываемыми в модели. Составные концепты имеют два специальных атрибута: последовательность детей (концептов) и функция

конструктора (для обозначения, как дети соединяются). Если все дети некоторого концепта атомарны, то такой концепт называют *страницей* (рис. 2).

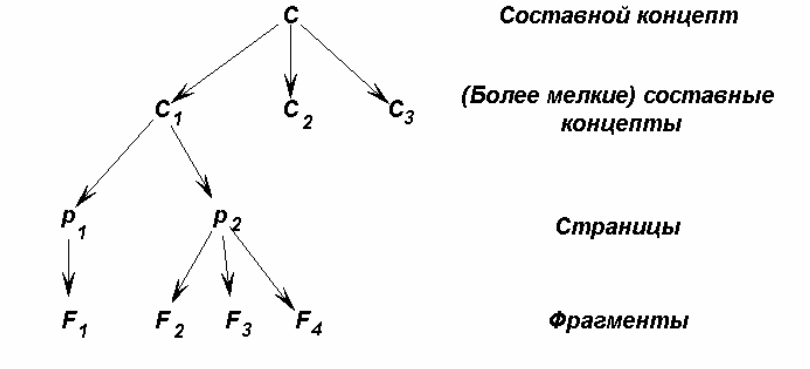


Рис. 2. Часть иерархии концептов

Межконцептные отношения представляют различные отношения между двумя или более концептами. Например, часто используются следующие типы бинарных отношений между парами концептов (рис. 3):

- тип *часть* (part-of) — это композиционное отношение, которое означает, что первый концепт представляет часть второго;
- тип *связь* (link) означает, что существует связь первого концепта со вторым (например, в виде гиперссылки);
- тип *предпосылка* (prerequisite) означает, что первый концепт должен быть предварительно изучен до изучения второго концепта;
- тип *блокиратор* (inhibitor) означает, что первый концепт не должен изучаться до тех пор, пока не будет изучен второй.

Межконцептные отношения могут иметь атрибуты. Например, приписывая некоторое вещественное число из интервала [0,1] тому или другому бинарному отношению, можно указать:

- для отношения типа *часть*, какую часть второго концепта представляет первый концепт;

- для отношения типа *предпосылки*, как много знаний о первом концепте должно быть у пользователя, чтобы информация о втором концепте становилась желаемой;
- для отношения типа *блокировки*, какую границу знаний о первом концепте должны превысить знания пользователя, чтобы стало нежелательным получение знания о втором концепте.

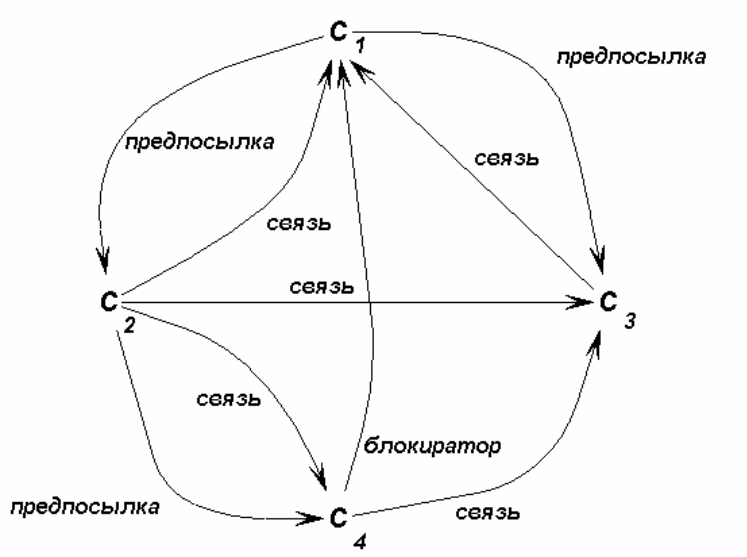


Рис. 3. Пример структуры межконцептных отношений

Можно выделить три типа моделей предметной области, различающихся по уровню сложности структуры [14].

- Самую простую структуру имеет модель *первого уровня*, являющаяся независимым множеством концептов (отсутствуют межконцептные отношения).
- Модель *второго уровня* (сетевая модель предметной области) предполагает наличие связей между концептами и представляет собой семантическую сеть, состоящую из концептов и межконцептных от-

ношений. Может существовать несколько типов концептов и межконцептных отношений.

- Модель *третьего уровня* (фреймовая модель предметной области) предполагает наличие у концептов внутренней структуры в виде множества атрибутов, при этом концепты различных типов могут иметь различные множества атрибутов.

Одной из самых важных функций модели предметной области является обеспечение структуры для представления знаний пользователя системой.

Адаптивные гипермедиа-системы можно подразделить на три основные группы в зависимости от метода организации связи между моделью предметной области (концептами) и гиперпространством системы (гипермедиа-страницами) [14].

- Самый простой метод — это *индексация страниц* концептами, относящимися к содержимому этих страниц; он может применяться даже для моделей предметной области первого уровня.
- Второй метод, похожий на предыдущий, это — *индексация фрагментов*: содержимое страницы разбивается на множество фрагментов, каждый из которых отдельно индексируется множеством концептов, относящихся к содержимому данного фрагмента. При разбиении на относительно небольшие фрагменты каждый фрагмент индексируется ровно одним концептом. Этот метод также можно использовать в моделях предметной области первого уровня.
- Третий метод (*прямая связь*) отличается от предыдущих методов тем, что для страниц не поддерживаются индексы; гиперпространство адаптивной гипермедиа-системы строится непосредственно исходя из структуры модели предметной области. Таким образом, каждый концепт модели представлен гипермедиа-страницей или гипердокументом, а отношения между концептами соответствуют гиперссылкам между страницами. Страница или документ, представляющий концепт, могут быть как статическими, так и динамическими: т.е. генерироваться налету, исходя из внутренней структуры концепта. Последний метод является самым мощным из всех вышеперечисленных, однако он требует использования модели предметной области второго, а лучше третьего уровня.

5.2. Модели пользователя

Модель пользователя адаптивной гипермедиа-системы предполагает *явное* представление знаний, предпочтений, целей, интересов, истории нави-

гации и других характеристик пользователя и служит для адаптации к пользователю различных аспектов адаптивной гипермедиа-системы. Модель пользователя состоит из именованных элементов, для которых хранится набор пар вида атрибут-значение (компонентов модели пользователя). На концептуальном уровне можно представлять модель пользователя в виде табличной структуры, в которой для каждого элемента хранятся значения атрибутов. Большинство элементов в модели пользователя представляют концепты модели предметной области. Некоторые другие элементы могут представлять различные аспекты пользователя, такие как цели, предпочтения, интересы или стереотипную классификацию (типа новичок, эксперт) и т.д. [35, 37, 96, 99, 100].

Можно классифицировать модели пользователей согласно следующим основным свойствам.

- *Способ получения информации: явный или неявный.* При явном способе система в явном виде запрашивает у пользователя информацию, необходимую для создания и обновления модели. Альтернативный подход, при котором система сама и незаметно для пользователя конструирует модель пользователя, отслеживая и анализируя взаимодействия с пользователем (например, историю навигации) для выведения различных предположений о пользователе, необходимых для построения его модели.
- *Степень специализации модели: общие или индивидуальные модели.* Общая модель предполагает наличие одной модели для всех пользователей системы, она используется в тех случаях, когда система предназначена для эксплуатации однородной группой пользователей. Индивидуальная модель принимает во внимание персональные особенности пользователей, и поддерживается своя для каждого пользователя. Стереотипная модель является способом комбинирования двух вышеперечисленных моделей: стереотип — набор характеристик, связанных друг с другом, определяющий некоторый класс пользователей.
- *Модифицируемость модели: статические или динамические модели.* В то время как статическая модель сохраняет характеристики пользователя, динамическая модель постоянно обновляется по мере получения новой информации в течение сеанса взаимодействия с пользователем.
- *Временная протяженность: краткосрочные или долгосрочные модели.* В отличие от краткосрочных моделей, долгосрочные модели

сохраняются от одного сеанса взаимодействия с пользователем до другого.

- *Метод использования модели: дескриптивные или прескриптивные модели.* Более традиционным является дескриптивное использование модели пользователя, когда модель пользователя — просто база данных, содержащая информацию о пользователе, у которой система может запрашивать текущие данные о пользователе. Прескриптивное использование модели предполагает, что система моделирует (имитирует) пользователя для проверки интерпретации ответа пользователем.

Модель пользователя хранит информацию об индивидуальном пользователе. Дискуссии о моделировании пользователя можно найти в [60, 65]. Различаются два основных типа техник, моделирующих пользователя: *моделирование перекрытий* и *моделирование стереотипного пользователя*.

5.2.1. Моделирование перекрытий

Оверлейное моделирование или *моделировании перекрытий* (overlay modeling) [47] чаще всего используется в интеллектуальных системах обучения для моделирования знаний, при этом знания пользователя описываются как подмножество знаний эксперта в данной области, отсюда сам термин «перекрытие» («оверлей»). Недостаток знаний обучающегося выводится посредством сравнения их со знаниями эксперта.

Для каждого концепта модели предметной области в модели знаний пользователя вычисляется и сохраняется некоторое значение (или несколько значений), оценивающее уровень знания пользователем этого концепта (оверлейная модель). Оверлейная модель знаний (overlay model) может быть представлена как множество пар «концепт — значения атрибутов».

Следующие атрибуты обычно используются для обозначения уровня знаний концепта пользователем [96, 99, 100].

- *«Значение знания»* (или просто *значение*) показывает уровень знания концепта пользователем. Все пары концепт-значение вместе формируют оверлейную модель, которая представляет «знания» пользователя. Некоторые адаптивные гипермедиа-системы используют булеву модель пользователя, в которой для каждого концепта определено, знает или не знает пользователь концепт. Другие используют небольшой набор значений, например «не известен», «изучен», «хорошо изучен» и «известен», или большой набор (например, проценты).

- Атрибут *«чтения»* показывает, просматривал (читал, изучал) ли пользователь какую-нибудь информацию (фрагмент, страницу или набор страниц) о концепте. В Web-системах атрибут чтения используется, чтобы генерировать различное представление для гиперссылок к посещенным и непосещенным страницам (по умолчанию это фиолетовый и синий цвета). Атрибут чтения может иметь булевы значения в некоторых адаптивных гипермедиа-системах, в то время как в других это может быть список времен доступа.
- Менее распространенный атрибут — это *«готовность для чтения»*, который показывает, готов ли пользователь для просмотра и изучения информации об этом концепте (это означает, например, что пользователь уже приобрел достаточное количество предварительно необходимых знаний).

Использование оверлейной модели особенно целесообразно, когда материал обучения может быть представлен в виде иерархии предварительных условий. Критичным в ее использовании является начальная оценка уровня знаний пользователя, поскольку количество наблюдений для достаточной оценки знаний должно быть небольшим.

Как уже было сказано выше, в рамках модели перекрытий предполагается, что знание пользователя составляет некоторое подмножество знания эксперта, и цель обучения состоит в расширении этого подмножества. Модель также предполагает, что пользователь не будет изучать того, чего не знает эксперт. В частности, не принимаются во внимание неправильные представления и заблуждения, изначально имеющиеся у пользователя, или приобретенные им в процессе обучения. Второй недостаток оверлейной модели заключается в том, что нет механизма для разграничения знаний, которые пользователь еще не приобрел, и знаний, которые еще не были ему представлены, что имеет смысл для стратегии обучения.

Дифференциальная модель (differential model) является расширением оверлейной модели, которое предполагает разделение знания предметной области на представленное и не представленное пользователю. Оверлейная модель применяется к тому знанию, которое уже представлено пользователю. В отличие от оверлейной, дифференциальная модель принимает во внимание неправильные представления и ошибки пользователя.

Пертурбационная модель (pertubation model) принимает во внимание знания, которыми обладает пользователь, но которые не представлены в модели знания эксперта предметной области. Пертурбационная модель расширяет модель эксперта добавлением библиотеки ошибок (bug library). Процесс ее создания может быть перечисляющим или порождающим. *Пе-*

речисляющий процесс составляет список всех возможных неправильных представлений с помощью анализа предметной области и ошибок, которые допускает пользователь. *Порождающий* метод пытается генерировать ошибки исходя из лежащей в основе теории познания. Оверлейная модель может быть применена поверх комбинированной модели эксперта и библиотеки ошибок. Как и для простой оверлейной модели, цель обучения — увеличить подмножество знания эксперта при исключении неправильных представлений.

5.2.2. Моделирование стереотипного пользователя

Стереотипное моделирование (stereotype modeling) — один из первых методов в области моделирования пользователя [81], который классифицирует пользователей по стереотипам. Предполагается, что пользователи, относящиеся к одному классу, имеют одни и те же характеристики. Классификация по стереотипам может быть проведена для каждого адаптационного признака. Этот метод целесообразно использовать в случаях, когда требуется быстрая, но не обязательно правильная оценка пользователя.

Можно определить стереотип как набор некоторых взаимосвязанных характеристик, присущих всем членам определенной подгруппы пользователей. Стереотипная модель различает несколько типичных (или «стереотипных») пользователей, например, «новичок», «средний», «эксперт» и т.п. В процессе моделирования пользователи образуют разные группы, причем каждый пользователь ассоциируется с одним стереотипом или несколькими. При этом предполагается, что пользователи, принадлежащие к одной группе, имеют одинаковые характеристики, присущие данной группе. Стереотипная классификация может быть использована для каждого аспекта адаптации.

Стереотипная модель может быть представлена как множество пар «стереотип—значение», где значением может быть «истина» или «ложь» (означающее, принадлежит ли пользователь данному стереотипу).

При использовании стереотипной модели иногда полезно различать два типа стереотипного моделирования: *фиксированное* моделирование и моделирование *по умолчанию*. При фиксированном моделировании обучаемые ассоциируются с предопределенными стереотипами на базе их представления. Стереотипное моделирование по умолчанию — более гибкий подход: в начале сессии обучаемые ассоциируются со стандартными стереотипами, но затем, по ходу процесса обучения, установки начальных стереотипов постепенно заменяются более индивидуальными установками.

Кроме характеристик подгрупп пользователей, стереотипы могут содержать так называемые триггеры (инициирующие условия), которые представляют ключевые характеристики, позволяющие идентифицировать пользователя как принадлежащего к соответствующей подгруппе пользователей. Триггеры могут относиться к текущим пользовательским характеристикам, эксплуатационным характеристикам (например, истории навигации), данным окружения (например, данным о конфигурации, оборудовании). Стереотипы применяются к пользователю, если они назначены «вручную» или если их триггеры совпадают с доступной информацией о пользователе (автоматическая классификация). В результате, все характеристики соответствующего стереотипа приписываются пользователю.

Суммируя вышесказанное, можно представить стереотип состоящим из следующих частей:

- множества инициирующих условий (триггеров), являющихся логическими выражениями, активирующими стереотип;
- множества условий отвода (ретракций), которые ответственны за деактивацию активного стереотипа;
- множества предположений (выводов) стереотипа, которые служат предположениями по умолчанию при связывании пользователя со стереотипом.

Можно сформулировать три этапа, которые нужно выполнить при разработке стереотипов:

- определение подгрупп пользователей: необходимо выделить внутри группы пользователей подгруппы, члены которых имеют некоторые однородные характеристики;
- идентификация ключевых характеристик: требуется определить небольшое число ключевых характеристик, позволяющих идентифицировать членов подгруппы пользователей (присутствие или отсутствие этих характеристик должно быть распознаваемо системой);
- представление в виде (иерархически упорядоченных) стереотипов: требуемые характеристики определенных групп пользователей должны быть формализованы в подходящей системе представления. Совокупность представленных характеристик подгруппы пользователей называется «стереотипом» этой подгруппы. Если содержимое одного стереотипа образует подмножество содержимого другого, может быть построена иерархия стереотипов, в которой содержимое стереотипа более высокого уровня наследуется стерео-

типом более низкого уровня и таким образом представлено один раз.

При использовании стереотипного моделирования может возникнуть следующая проблема: стереотипы могут быть так специализированы, что будут состоять только из одного пользователя, или пользователь вообще не сможет быть классифицирован.

Можно определить четыре модели согласно типу организации связи между стереотипами: луковая (многоуровневая) модель, летисная модель, многоядерная летисная модель, ориентированный ациклический граф.

- *Многоуровневая* (или *луковая*) *модель* (onion model) — это иерархическая модель, в которой содержимое стереотипов линейно упорядочено по отношению быть подподмножеством. Пользователи, принадлежащие к некоторому стереотипу *S*, наследуют все знание, ассоциированное с более общими стереотипами, чем *S* (наследование без исключений). В качестве примера можно привести классификацию пользователей, в которой «новички» знают подмножество концептов *A*; «продвинутые» пользователи имеют знание новичков, но они также знают концепты из подмножества *B*; наконец, «эксперты» знают концепты из *A, B* и *C*.
- *Летисная* (или *салатная*) *модель* (lettuce model) характеризуется наличием стереотипа-ядра, содержимое которого является подмножеством содержимого всех других стереотипов, которые в остальном могут быть независимы друг от друга. Эта модель часто используется для представления знаний пользователей программных систем подобных командам UNIX: пользователи владеют только относительно небольшим множеством основных команд и рядом функционально связанных команд (например, для управления файловой системой, электронной почтой и т.д.).
- *Многоядерная летисная* (или *многоядерная салатная*) *модель* (multikernel lettuce model) является обобщением летисной модели, в котором допускается существование нескольких ядер, являющихся пересечениями некоторых стереотипов.
- *Ориентированный ациклический граф* (DAG) является обобщением многоядерной модели, в котором допускается существование общих частей у двух или более ядер, которые также представляются ядрами, и т.д. В результате получается иерархия стереотипов, в которой каждый узел может иметь больше одного узла высшего уровня.

5.3. Модели адаптации

Модель адаптации описывает, как должна происходить адаптация в зависимости от моделей пользователя и предметной области. Она состоит из правил адаптации, которые формируют связь между моделью предметной области и моделью пользователя и определяют представление генерируемой информации и обновление модели пользователя [35, 37, 99, 100].

Обычно правило содержит две основные части: *условие*, при котором происходит срабатывание данного правила, и *трансформация* — описание того действия, которое задается данным правилом. Условие может предполагать возникновение некоторого внешнего события (такие правила называются ECA-правилами, в отличие от CA-правил, в которых условие — это просто логическое выражение), например, обращения к странице, а также истинность некоторого логического выражения, построенного над значениями атрибутов из моделей пользователя и предметной области и проверяемого в те моменты, когда событие возникает. Действие может состоять в модификации значений атрибутов в модели пользователя или в присваивании объекту спецификации представления. Также, в правиле может быть «фаза» выполнения, указывающая на момент времени, в который должно применяться правило: до или в течение генерации (фаза “pre”) и после генерации страницы (фаза “post”). Основанием для наличия двух стадий выполнения является то, что сначала может потребоваться осуществить некоторую адаптацию, основываясь на «текущем» состоянии модели пользователя (фаза “pre”), а затем модифицировать модель пользователя после генерации представления страницы (фаза “post”). Также в правиле может быть определено, может ли оно инициировать запуск других правил или нет.

Как показано в работе [8], многие из реальных приложений используют такие ECA-правила, в которых логические выражения принимают истинные значения в точности тогда, когда возникают события, составляющие условия данных правил; такие правила получили название квази-CA-правил.

Все правила адаптации подразделяются [98] на *родовые* или *обобщенные* правила (generic rules) и *специфические* или *конкретные* правила (specific rules). В отличие от обобщенных правил, которые применимы ко всем концептам и всем межконцептным отношениям некоторого заданного типа и используют связанные переменные для представления в них концептов и межконцептных отношений, специфические правила описывают преобразования для конкретных концептов, множеств концептов и межконцептных отношений и не используют переменных. Специфические правила имеют

приоритет над обобщенными правилами, и таким образом, они используются для определения исключений в общих правилах.

В качестве примера можно привести два простых правила, написанных с использованием произвольно выбранного синтаксиса.

Например, следующее правило определяет, что при обращении к странице для соответствующего концепта в модели пользователя устанавливается атрибут «чтение» равным истине в фазе “post”:

<access(C) => C.read:= true; post; true>

Правило также утверждает, что оно запустит другие правила, которые имеют атрибут «чтения» (C.read:= true) в своей левой части.

Другой пример. Следующее правило выражает, что когда пользователь обращается к странице, определяющей концепт, «готовый для чтения», то значение знания для этого концепта становится «изученным» в фазе “pre”:

<(access(C) & C.ready-to-read = true) => C.knowledge-value:=learned; pre; true>

Одной из важных проблем, возникающих при построении модели адаптации, является обеспечение таких свойств систем правил адаптации, как нетеровость и завершаемость [5], позволяющих не следить за порядком применения правил адаптации (с помощью механизма адаптации).

6. ПРИМЕРЫ АДАПТИВНЫХ ГИПЕРМЕДИА-СИСТЕМ

Много интересных адаптивных гипермедиа-систем для обучения было разработано в начале 90-х годов прошлого столетия. Интерес, вызванный развитием дистанционного Web-обучения, заметно усилил эти исследования, увеличив количество и типы разрабатываемых систем. Все первые системы были в значительной степени лабораторными, построенными исключительно для исследования тех или иных методов адаптации в контексте обучения. В отличие от них, ряд более современных систем обеспечивает среды и авторизованные инструменты для разработки Web-курсов. Появление большого числа авторизованных инструментов демонстрирует зрелость адаптивной обучающей гипермедиа и является ответом на Web-спровоцированным запросом на адаптивных к пользователям дистанционных учебных курсов.

Выбор Web в качестве платформы разработки адаптивных гипермедиа стал стандартом. Этот выбор дал долгую жизнь ряду основанных на Web-адаптивных учебных гипермедиа-систем, разработанных до 1996 г., таких

как ELM-ART, InterBook и 2L670. Эти системы были существенно модифицированы после 1996, расширены большим числом новых технологий и использованы для ряда экспериментальных исследований. Не удивительно, что почти все адаптивные учебные гипермедиа-системы, разработанные после 1996 г., являются Web-системами.

6.1. ELM-ART

Система ELM-ART [22] и ее «наследники» ELM-ART II [95] и INTERBOOK [21] были одними из первых адаптивных гипермедиа-систем, используемых в Интернете. Они основаны на отдельной системе ELM-PE [93], вводном курсе по программированию на LISP. Ее авторы используют *эпизодическую модель ученика* [94] (ELM) для диагностики полных и неполных решений проблем. Эпизодическая модель хранит информацию о пользователе в виде коллекции *эпизодов*. Эти эпизоды могут сравниваться с прецедентами в обучении, основанном на прецедентах [85]. Для построения модели ученика код написанной им программы анализируется в терминах области знаний, с одной стороны, и описания задачи, с другой. Эта диагностика дает в итоге дерево вывода понятий и правил, которые ученик мог использовать при написании программы.

В ELM-ART понятия связаны друг с другом посредством предварительных условий и результатов. Таким образом, строится понятийная сеть. Наблюдения за пользователем производятся посредством мониторинга посещенных страниц: понятие, соответствующее посещенной странице, помечается в понятийной сети как известное пользователю. Для аннотации ссылок авторы используют метафору светофора. Красный кружок обозначает страницы, для изучения которых пользователю недостает знаний; зеленый обозначает страницы, на которые предлагается обратить внимание, и т.д. ELM-ART также содержит интерактивные примеры, которые могут быть оттранслированы LISP-компилятором через Интернет.

Система ELM-ART II была разработана для перевода обычных учебников в электронные. Она улучшает представление знаний, имевшееся в ELM-ART. Понятийная сеть иерархически организована в лекции, секции, подсекции и конечные страницы. Каждый элемент понятийной сети имеет слот, содержащий текст для страницы и информацию о связи этой страницы с другими элементами. Статические слоты содержат предварительные требования, родственные понятия и результат. Конечные страницы содержат тестовые слоты. Страницы с задачами имеют определенный слот для хранения описания программной задачи. Индивидуальная модель ученика хранит

посещенные страницы, решенные тесты и решенные программные задачи, помечая соответствующие понятия в понятийной модели как «известные».

Прямое руководство осуществляется с помощью кнопки «следующий лучший», подсказка предоставляется с помощью нахождения наиболее подходящего примера из индивидуальной истории обучения, основываясь на диагностике программного кода, использованного учеником в своих решениях.

Системы способны делать заключения о знаниях пользователей, основываясь на помеченных понятиях в понятийной модели. Все предварительные требования, считающиеся необходимыми для известных пользователю понятий, также рекурсивно помечаются как известные.

6.2. INTERBOOK

В проекте INTERBOOK [22] электронные учебники создаются на базе иерархически структурированных файлов MS-Word. Для этого должны быть выполнены такие операции, как создание списка понятий предметной области, структурирование и аннотирование страниц с выдаваемыми результатами и предварительными требованиями, трансляция в HTML и разбор информации на структуры LISP.

INTERBOOK использует и модель предметной области, и модель пользователя. Глоссарий и учебник основаны на модели области. Глоссарий есть визуализированная понятийная сеть. Структура глоссария напоминает дидактическую структуру знаний данной области. Каждый пункт глоссария соответствует одному из понятий области. Вдобавок к этому каждая запись в глоссарии содержит ссылки на все разделы книги, которые используют это понятие.

Каждый элемент учебника помечен некоторыми понятиями модели области. Эти понятия играют различную роль. Некоторые из них описывают *результаты* — знания, которые пользователь получил после изучения страницы, другие описывают *предварительные* условия, или знания, необходимые для понимания этой страницы.

INTERBOOK поддерживает адаптивную аннотацию ссылок, используя метафору светофора. Подсказка, основанная на предварительных условиях, осуществляется с помощью предоставления аннотированного списка страниц, содержащих предварительно необходимую информацию.

Упорядочение страниц производится в три этапа. Вначале система вычисляет общее количество «очков», отражающих предполагаемое состояние знаний, для каждого понятия. Основываясь на этих цифрах, система решает,

хорошо ли усвоено понятие или нет. Затем система решает, какие страницы содержат рекомендованную обучающую информацию либо недостающие предварительные знания. Ссылки на понятия и разделы разных образовательных состояний аннотированы различными пиктограммами. Наконец, система выбирает наиболее оптимальную страницу среди всех доступных страниц, на которых вводятся неизученные понятия и все предварительные условия которых уже известны. Всем страницам присваивается определенный приоритет их представления, который выводится из базового значения в соответствии с состоянием знаний о требуемых и вводимых понятиях.

6.3. INTERBOOK: адаптивный интерфейс

Подход к реализации адаптивного пользовательского интерфейса для проекта INTERBOOK предложен в [20]. Модель области и модель ученика используются для его представления. Каждое свойство интерфейса рассматривается как понятие области, а каждая подсказка — это обучающий элемент. Применяя алгоритм упорядочения страниц, подобный описанному выше, генерируются последовательность свойств интерфейса, которые нужно изучить, и последовательность подсказок, которые нужно показать пользователю.

6.4. PT

PT (персонализированная текстовая система) — это учебник по изучению программирования на языке C [61, 62]. Она использует обычную книгу о C и генерирует на ее основе гипермедиа-систему. Курсом, который поддерживается PT, является курс по программированию на C для программистов на Pascal.

PT использует стереотипную пользовательскую модель целевой аудитории (программистов на Pascal) в дополнение к индивидуальной модели. Стереотип дает определенные значения компонентов знания, инициализируя таким образом модель пользователя. В индивидуальной модели хранятся значения знаний индивидуального пользователя во время его работы с PT.

Для того чтобы сделать возможной адаптацию, PT использует сходство языков Pascal и C при представлении информации пользователю. Команды препроцессора добавляются в «сырой» HTML-код страницы из модели пользователя, например:

#define PASCAL 3,

#define active-learner 1,

#if PASCAL > 2.

Команды «if» препроцессора на HTML-странице используются для контроля того, какие именно части страницы передаются пользователю. Эта же техника препроцессинга используется для выбора ссылок.

6.5. PUSH

Проект PUSH (подсказка, настроенная на пользователя и план) ставит целью разработку и тестирование интеллектуальных справочных решений к задачам поиска информации [53]. Пользователь может вводить вопросы, в том числе повторные, или перемещаться по графическому представлению предметной области. Когда пользователь сформулировал вопрос, он может воздействовать на ответ системы, открывая и закрывая подсекции, изменяя графику или выбирая последующие предлагаемые вопросы.

Знания о предметной области моделируются isA-иерархией. Идея системы состоит в том, чтобы заготовить вопросы, которые могут возникнуть у пользователя при чтении документа. Вопросы затрагивают родственные понятия и знания. Так как комбинаторная конструкция всех возможных вопросов, основанных на понятиях и их взаимосвязях в isA-иерархии, будет слишком сложной и содержащей слишком много информации, множество возможных вопросов ограничено понятиями, содержащимися на странице, и повторными вопросами для этих понятий. Система реализует подход к запросам, основанный на правилах. Как только пользователь верифицирует каждую из своих задач, выбирая некоторые из предлагаемых ссылок, система получает практически точную информацию о знаниях пользователя.

6.6. AHA

Бесплатный Web-курс по *гипермедиа-структурам и системам* [23, 33] реализован в системе AHA. AHA (адаптивная гипермедиа-архитектура) может быть использована для генерации условных тестов, а также для адаптации ссылочной структуры с помощью удаления, скрытия и аннотации ссылок. Команды препроцессора на HTML-страницах используются CGI-скриптами для фильтрации содержания фрагментов страницы, таким способом реализуется адаптация на уровне содержания. Та же препроцессорная техника используется для адаптации на уровне ссылок.

AHA! Версии 1.0 [38] делает возможным создание сайтов, в которых ссылки могут быть спрятаны или аннотированы, а фрагменты — включены или опущены, основываясь на правилах адаптации и модели пользователя с очень гибкой структурой (продукционные правила работают в структуре

концептов и атрибутов). AHA! 1.0 — инструмент широкого назначения в том смысле, что он не навязывает один стиль презентации и что адаптация может быть основана на произвольных событиях (arbitrary events) и зависимостях между концептами (в отличие от, например, обучающих систем, в которых предполагается монотонный процесс усвоения знаний посредством чтения страницы). Однако в данной версии AHA! все еще довольно простая система. Она работает только с правилами, созданными вручную, и условное включение фрагментов — единственный тип адаптации содержания, который она поддерживает.

В AHA! Версии 2.0 авторы направили силы на создание более богатой структуры модели пользователя, предметной области и модели адаптации с более развитыми *требованиями* и правилами *генерации*. Новые особенности системы, разработанные авторами, основаны на ссылочной (reference) модели AHAM [37], которая является расширением хорошо известной ссылочной модели Dexter для гипермедиа [50]. AHA! отличается от ссылочной модели AHAM [37] тем, что в ней не отделена *модель предметной области* от *модели адаптации*. В AHA! автор определяет *концепты* наряду с *требованиями*, которые определяют условия, когда пользователь "готов" перейти к концепту, и *правилами генерации*, которые определяют, как поведение просмотра (browsing behavior) пользователя интерпретируется при обновлении модели пользователя. *Правила генерации* являются *продукционными правилами (condition-action rules)*, как принято в теории активных баз данных [8]. AHA! 2.0 поддерживает хранение структуры концептов в XML-файлах или в базе данных mySQL.

6.7. OLAE, POLA

Система OLAE [69] нацелена на дифференцированное и надежное определение знаний обучающегося в области физики. Сеть Байеса [77] строится для каждой проблемы, над которой работает пользователь. Эта сеть отражает, среди прочего, вероятность того, что пользователь вводит определенные уравнения, если он знает соответствующие правила. Таким образом, система использует ретроспективную диагностику, называемую *отслеживанием знаний*. POLA [27] разработана для *отслеживания моделей*: она может вызываться многократно во время процесса решения проблемы. POLA строит базовую сеть Байеса с помощью приращения узлов, добавляя их каждый раз, когда обучающийся производит наблюдаемое действие.

6.8. POKS

Система POKS [40] основана на когнитивной теории структур знания. Она строит сеть выводов на элементах знаний из небольшого образца из множеств данных пользователя и использует эту «наведенную» сеть для доступа к состоянию знаний, обладая ограниченным количеством наблюдений или вопросов. Система применяется для адаптированного построения пользовательских интерфейсов. Представлено моделирование перекрытий.

6.9. EPI-UMOD

Система EPI-UMOD [83] реализует модель стереотипного пользователя, основанную на сетях Байеса. Она использует отдельную сеть Байеса для каждого стереотипа, в которой реализованы специальные условные зависимости между единицами знаний. Каждый атрибут этого стереотипа представляет собой утверждение, что пользователю известно определенное понятие.

6.10. HYDRIVE

Система HYDRIVE [70] нацелена на обучение технического и летного персонала в области функционирования гидравлических систем самолета F15. Основной упор делается на понимание системы и стратегии разрешения проблем, а не на оптимизацию действий, которые необходимо предпринимать в заданном пункте проблемы. Авторы используют иерархическую модель необходимых способностей обучающихся и строят сеть Байеса для всего сценария приложения. Однако эта сеть модифицируется только частично.

6.11. Система KBS-гиперкниг

Целью системы KBS-гиперкниг [51, 52] является построение структуры для разработки и поддержки открытых адаптивных гипермедиа-систем в Интернете. Она реализует обучение, основанное на проектах [85].

В системах баз знаний (KBS, knowledge-based systems) понятия связаны друг с другом на базе понятийной модели гиперкниги. Наблюдения за пользователями производятся, когда они выполняют некоторые проекты в библиотеке проектов гиперкниги.

Каждый элемент гиперкниги индексирован некоторыми понятиями из области знаний. Строится отдельная модель знаний, содержащая понятия

знаний из предметной области и зависимости, их обучающие. Таким образом, сам по себе гипертекстовый документ не содержит никакой информации о предварительных требованиях или результатах.

Генерируется глоссарий, содержащий понятия из модели знаний. Для каждого пункта глоссария генерируются ссылки на примеры, на элементы гиперкниги и на страницы других электронных книг, доступных в Интернете.

KBS использует метафору светофора для адаптивной аннотации. Алгоритм упорядочения страниц генерирует последовательность чтения в соответствии с целями и знаниями пользователя. Чтобы помочь пользователю проложить собственный путь через гиперкнигу, система также генерирует следующий обучающий шаг путем сравнения текущего состояния знаний пользователя с состоянием знаний, которое он должен иметь после завершения работы с книгой.

KBS поддерживает обучение, основанное на целях. Пользователи могут определять собственные цели обучения или запрашивать следующую цель у системы. Для каждой из этих целей генерируется последовательность чтения, содержащая необходимые знания (как предварительные требования, так и текущее состояние знаний) для достижения цели. Вдобавок к этому, выбираются подходящие проекты и предлагается информационный индекс, содержащий как документы из самой гиперкниги, так и другие доступные в Интернете материалы.

Выбор подходящих проектов основывается на алгоритмах, которые отражают как предварительные знания, необходимые для выполнения проекта, так и то, насколько хорошо проект соответствует текущей цели обучения.

KBS также адаптируется к различным скоростям обучения пользователей, поддерживая эту разновидность обучения, основанного на целях. Пользователи могут сами определять, сколько и чего именно они хотят изучить на следующем шаге. Если система замечает, что пользователь овладевает проектами повышенной сложности достаточно успешно, она модифицирует свою оценку данного пользователя в отношении пройденных тем, а также предварительных знаний. Таким образом, пользователь сможет перейти к другим, более продвинутым темам. Если система обнаруживает, что пользователь недостаточно знаком с некоторой темой, она предлагает сходные примеры или проекты, содержащие минимальное количество новой информации.

Техника реализации, использованная в KBS, представляет собой сеть Байеса над полной моделью предметной области. Когда бы ни были произ-

ведены наблюдения над конкретным пользователем, они далее распространяются по всей сети.

Важное внимание в KBS уделяется расширяемости системы по отношению к Интернету. Чтобы иметь возможность создавать *открытые* адаптивные гипермедиа-системы, выбранный в KBS подход к индексированию позволяет обрабатывать каждую информационную единицу одинаково независимо от ее происхождения. Таким образом, HTML-страницы, найденные в Интернете, могут быть интегрированы в систему таким же образом, как и документы, размещенные в библиотеке гиперкниги.

6.12. TILE

Система TILE разрабатывается [63] как интегрированная адаптивная система для дистанционного обучения с моделью пользователя, основанной на клиент-серверном подходе. TILE поддерживает модели отдельных студентов и групп студентов.

Индивидуальная модель содержит четыре основных составляющих: глобальные предпочтения; специфическое представление содержания, связанное с предпочтениями обучаемого; компетентность в предметной области; история работы студента. Она используется системой для обеспечения адаптивного навигационного управления, выбора граннулированности содержания предметной области, обеспечения основанных на контекстах экскурсий к другим единицам обучения, нахождения аналогий с ранее изученным материалом, создания прямых ссылок на ранее изученный материал, обеспечения динамических сообщений и обратных связей.

Система суммирует общее поведение студентов и предпочтений для создания и сопровождения групповой модели. Студенты, использующие систему, разбиваются на различные группы, соответствующие их поведению и компетентности в предметной области, которые отражаются в модели определенных студенческих групп. Структура групп многомерна, и один студент может принадлежать одной или нескольким таким группам.

Групповая и частично индивидуальная модели размещаются на сервере, в то время как у клиента находится только частично индивидуальная модель.

Функционально система TILE содержит следующие компоненты: модуль студенческой модели; модуль предметной области (представление базы знаний); модуль эксперта, поддерживающий единицы обучения (лекции, примеры, тесты и т.д.), каталог ошибок и объяснения для элементов

учебного плана; модуль обучения, содержащий действия по адаптации и обработке студенческих действий.

6.13. SAC

Система SAC [26] — система регулируемого адаптивного обеспечения курсов, основанная на ссылочной модели AHAM [37]. Структура курса, материалы и цели обучения представляются абстракциями вершин курса, единиц курса и материала курса, как это описано в модульной системе обучения MTS [91].

Курсовое обеспечение реализуется с помощью архитектуры модели трехярусного приложения. Первый ярус реализуется с помощью браузера, который связан с представлением обеспечения курса. Средний уровень реализуется и обеспечивается комбинацией Web-сервера и сервера приложений. Сервер приложений генерирует XML, DHTML и клиентский Джаваскрипт, динамически используя модельную информацию, сохраняемую в третьем ярусе системы базы данных.

Адаптивные возможности SAC обеспечиваются адаптивным агентом, который состоит из следующих трех частей: предвходовый адаптивный механизм, онлайновый индивидуальный адаптивный механизм и адаптивный механизм предметной области.

6.14. WebVT

Система WebVT [90] является адаптивной сетевой интеллектуальной программой обучения языку с помощью компьютера, которая предназначена для пассивного овладения английским языком лицами, для которых он не является родным языком.

Система объединяет возможности интеллектуальных систем обучения и адаптивных гипермедиа-систем. Она использует комбинацию стереотипного моделирования и моделирования перекрытий для инициализации студенческой модели. Эта модель в дальнейшем уточняется в процессе наблюдения за студентом, когда он работает с системой. Результирующая модель студента используется для аннотаций ссылок на топики, представляемые студенту. Кроме того, она также используется в процессе диагностики ошибок и адаптации обратной связи и советов, выдаваемых студенту.

6.15. ITS

Интеллектуальная система обучения ITS [79] предназначена для обучения использованию новых технологий преподавателей высших учебных заведений. Она предлагает единицы курса, покрывающие потребности пользователей с различными уровнями знаний и различными характеристиками.

Система настраивает представление учебного материала на различные пользовательские потребности с использованием технологий искусственного интеллекта для спецификации каждой пользовательской модели и принятия педагогических решений. Это достигается с помощью экспертной системы, которая использует формализм гибридного представления знаний, интегрирующий символические правила с нейровычислениями.

ЗАКЛЮЧЕНИЕ

Системы дистанционного обучения в настоящее время активно исследуются и развиваются. Выгоды сетевого обучения ясны: аудиторная и платформенная независимости. Сетевое обучающее программное обеспечение, один раз установленное и обслуживаемое в одном месте, может использоваться в любое время и по всему миру тысячами учащихся, имеющих любой компьютер, подключенный к Интернету. Тысячи программ сетевого обучения и других образовательных приложений стали доступны в сети за последние годы. Проблема состоит в том, что большинство из них являются не более чем статичными гипертекстовыми страницами. Поэтому целью ведущихся исследований является развитие продвинутых сетевых образовательных приложений, которые смогли бы предложить некоторое количество адаптивности и интеллектуальности. Эти свойства стали особенно важны для Web-систем дистанционного обучения с тех пор, как обучаемые стали обучаться в основном самостоятельно (обычно из дома). Интеллектуальное и личное содействие, которое могут дать учитель или студент-сокурсник при обычном (аудиторном) обучении, при дистанционном обучении нелегко достижимо. Адаптивность важна для программного обеспечения дистанционного обучения еще и потому, что оно должно использоваться намного более разнообразным множеством студентов, чем любое “однопользовательское” учебное приложение. Сетевое программное обеспечение, разработанное для одного класса пользователей (с одним складом ума), может совсем не подойти другим обучаемым.

В статье мы рассмотрели методы и средства адаптации, используемые в современных обучающих Web-системах. Не вызывает сомнения, что как адаптивные, так и интеллектуальные технологии могут повысить качество образовательных систем, используемых в сети. Например, адаптивное представление может сделать представление учебного материала более удобным для использования. Адаптивная поддержка в навигации и адаптивное построение последовательности могут использоваться как для повсеместного контроля над курсом обучения, так и для помощи студенту в выборе наиболее подходящих тестов и заданий. Поддержка в решении задач и интеллектуальный анализ решений могут значительно улучшить процесс выполнения заданий студентами, обеспечивая их интерактивной и интеллектуальной обратной связью на фоне значительного уменьшения нагрузки на преподавателя. Технологии подбора моделей могут улучшить как управление дистанционными курсами, так и взаимодействие между обучаемыми и преподавателями.

Однако, несмотря на имеющий потенциал, адаптивные и интеллектуальные технологии пока еще не нашли себе место в «реальной» виртуальной аудитории, т.е. пока их нет в массово используемых средствах обучения. Большинство систем, обсуждавшихся выше, — это типичные «лабораторные» (экспериментальные) системы, которые вообще не имели практики использования в реальных дистанционных курсах. Небольшая часть систем (в основном это системы из семейств ELM-ART и AHA) имеют небольшую практику использования, скорее экспериментального характера. В то же время ни одна из существующих коммерческих и «университетских» систем обучения, поддерживающих сотни дистанционных курсов, не использует ни адаптивные, ни интеллектуальные технологии.

Здесь авторы разделяют позицию П. Бруселовского [11], заключающуюся в следующем. Сетевое образование само по себе относительно молодо. До настоящего времени различные компании, производящие системы для сетевого образования, были способны конкурировать на рынке, создавая простые не- адаптивные системы. Однако большое количество экспериментальных систем демонстрирует явные преимущества адаптивных и интеллектуальных технологий. По ходу увеличения конкуренции на рынке образовательных Web-систем “быть адаптивной” или “быть интеллектуальной” станет важным фактором для завоевания покупателей. Традиционные компании создания систем сетевого образования начнут использовать адаптивные и интеллектуальные методы. Команды исследователей с серьезным опытом использования адаптивных и интеллектуальных технологий получат возможность для вывода своих технологий на рынок. Первыми техно-

логиями для использования в коммерческих системах, вероятно, будут технологии построения последовательностей (последо-вательность страниц и последовательность вопросов), так как они вполне подходят к существующей структуре обучающих систем дистанционного образования. Потом наступит очередь адаптивной поддержки в навигации и подбора модели. Технологии поддержки в решении задач будут оставаться на уровне исследований дольше, хотя мы можем ожидать появления небольших Web-тренажеров, предназначенных для поддержки изучения тех или иных частей некоторых предметов. Можно надеяться, что в ближайшие годы появится несколько примеров адаптивных и интеллектуальных систем коммерческого уровня, так же как и множество новых и перспективных разработок экспериментальных систем для дистанционного обучения.

СПИСОК ЛИТЕРАТУРЫ

1. Волянская Т.А. Виртуальный музей истории информатики в Сибири: модель предметной области и модель пользователя // Новые информационные технологии в науке и образовании. — Новосибирск, 2003. — С. 124–146.
2. Волянская Т.А. Методы и технологии адаптивной гипермедиа // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 38–68.
3. Зайцева Л.В. Методы и модели адаптации к учащимся в системах компьютерного обучения // Educational Technology & Society. — 2003. — Vol. 6, N 4. — P. 204–211.
4. Касьянов В.Н. О работе 16 Всемирного компьютерного конгресса ИФИП // Поддержка супервычислений и интернет-ориентированные технологии. — Новосибирск, 2001. — С. 9–30.
5. Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение. — СПб.: БХВ-Петербург, 2003. — 1104 С.
6. Albrecht D., Zukerman I., Nicholson A., Bud A. Towards a Bayesian model for keyhole plan recognition in large domains // Proc. of the Sixth Internat. Conf. on User Modeling (UM97). — Sardinia, 1997.
7. Alpert S.R., Singley M.K., Fairweather P.G. Deploying intelligent tutors on the Web: An architecture and an example // Intern. J. of Artificial Intelligence in Education. — 1999. —Vol. 10. — P. 183–197.
8. Baralis E., Widom J. An algebraic approach to static analysis of active database rules // ACM Transactions on Database Systems. — 2000. — Vol. 20, N 1. — P. 269–332.
9. Beaumont J. User modelling in the interactive anatomy tutoring system ANATOM-Tutor // User Modeling and User Adapted Interaction. — 1994. — Vol. 4, N 1. — P. 21–45.

10. Berners-Lee T. World Wide Web: An illustrated seminar. Held as an Online Seminar in 1991. http://www.w3.org/pub/WWW/Talks/General.html.
11. Brusilovsky P. Adaptive and Intelligent Technologies for Web-based Education // Konstliche Intelligenz. Special Issue on Intelligent Systems and Teleteaching. — 1999. — N 4. — P. 19–25.
12. Brusilovsky P. Adaptive Educational Systems on the World-Wide-Web: A Review of Available Technologies // Proc. of Workshop «WWW-Based Tutoring» at 4th International Conference on Intelligent Tutoring Systems (ITS'98). — San Antonio, 1998.
13. Brusilovsky P. Adaptive Hypermedia // User Modeling and User-Adapted Interaction. — 2001. — Vol 11. —P. 87–110.
14. Brusilovsky P. Adaptive hypermedia, an attempt to analyze and generalize. // Lect. Notes. Comput. Sci. — 1996. — Vol. 1077. — P. 288–304.
15. Brusilovsky P. Efficient techniques for adaptive hypermedia // Lect. Notes. Comput. Sci. — 1997. — Vol. 1326. — P. 12–30.
16. Brusilovsky P. Methods and techniques of adaptive hypermedia // Adaptive Hypermedia and Hypermedia. —Dordrecht: Kluwer Academic Publishers, 1998. — P. 1–43.
17. Brusilovsky P. Methods and techniques of adaptive hypermedia // User Modeling and User-Adapted Interaction. — 1996. — Vol 6, N 2-3. — P. 87–129.
18. Brusilovsky P., Cooper D. W. Domain, Task, and User Models for an Adaptive Hypermedia Performance Support System. // Proc. of 2002 Internat. Conf. on Intelligent User Interfaces. — San Francisco, CA, 2002. — P. 23–30.
19. Brusilovsky P., Pesin L. ISIS-Tutor: An intelligent learning environment for CDS/ISIS users // Proc. of the interdisciplinary workshop on complex learning in computer environments (CLCE'94). — Joensuu, 1994.
20. Brusilovsky P., Schwarz E. User as student: Towards an adaptive interface for advanced Web-based applications // Proc. of the Sixth Internat. Conf. on User Modeling (UM97). — Sardinia, 1997.
21. Brusilovsky P., Schwarz E., Weber G. A tool for developing adaptive electronic textbooks on WWW // Proc. of World Conf. of the Web Society (WebNet'96). — Boston, 1996.
22. Brusilovsky P., Schwarz E., Weber G. ELM-ART: An intelligent tutoring system on world wide web // Lect. Notes Comput. Sci. — 1996. — Vol. 1086. — P. 261–269.
23. Calvi L., De Bra P. Improving the usability of hypertext courseware through adaptive linking // Proc of the Eighth ACM Internat. Hypertext Conf. — Southampton, 1997.
24. Carver C.A., Howard R.A., Lavelle E. Enhancing student learning by incorporating student learning styles into adaptive hypermedia // Proc. of World Conf. on Educational Multimedia and Hypermedia (ED-EDIA'96), — Boston, MA, 1996. — P. 118–123.
25. Carver C.A., Howard R.A., Lavelle E. Enhancing student learning by incorporating student learning styles into adaptive hypermedia //Proc. of World Conf. on Educa-

tional Multimedia and Hypermedia (ED-MEDIA'96). — Boston, 1996. — P. 118–123.
26. Chan A.T.S., Chan S.Y.C., Cao J. SAC: a self-paced and adaptive courseware systems // Proc of IEEE Intern. Conf. on Advanced Lerning Technologies. — IEEE Computer Society, 2001. — P. 78–81.
27. Conati C., Gertner A.S., Van Lehn K., and Druzdzel M. J. Online student modeling for coached problem solving using bayesian networks // Proc. of the Sixth Internat. Conf. on User Modeling (UM97). — Sardinia, 1997.
28. Dagum P., Galper A., Horvitz, E. Dynamic network models for forecasting // Proc of the Eighth Conf. on Uncertainty in Artificial Intelligence. — San Mateo, 1992. — P. 41–48.
29. Danielson R. Learning styles, media preferences, and adaptive education // Proc. of Workshop "Adaptive Systems and User Modeling on the World Wide Web" at 6th Internat. Conf. on User Modeling (UM97) — Sardinia, 1997. — P. 31–35.
30. De Bra P. Adaptive Hypermedia on the Web: Methods, techniques and applications // Proc. of the AACE WebNet'98 Conf. — Orlando, 1998. — P. 220–225.
31. De Bra P. Design issues in adaptive Web-site development // Proc. of the 2nd Workshop on Active Systems and User Modelling on WWW. — Eindhoven, 1999. — P. 29–39.
32. De Bra P. Hypermedia structures and systems: Online Course at Eindhoven University of Technology, 1997. `http://wwwis.win.tue.nl/2L690/`.
33. De Bra P. Teaching hypertext and hypermedia through the Web // Proc. of of the Web Society (WebNet'96). — San Francisco, 1996.
34. De Bra P., Aerts A., Smits D., Stash N. AHA! Version 2.0, More Adaptation Flexibility for Authors // Proc. of the AACE ELearn'2002 Conf. — 2002. — P. 240–246.
35. De Bra P., Brusilovsky P., Houben G.-J. Adaptive Hypermedia: From Systems to Framework // ACM Computing Surveys. — 1999. — Vol. 31, N 4. — Article N 12.
36. De Bra P., Calvi L. AHA! An open Adaptive Hypermedia Architecture // The New Review of Hypermedia and Multimedia. — 1998. — P. 115–139.
37. De Bra P., Houben G.J., Wu H., AHAM: A Dexter-based Reference Model for Adaptive Hypermedia // Proc. of ACM Hypertext'99. — Darmstadt, 1999. — P. 147–156.
38. De Bra P., Stash N. AHA! A General-Purpose Tool for Adaptive Websites // Proc. of the World Wide Web Conf. — Lect. Notes Comput. Sci. — 2002. — Vol. 2347. — P. 381–384. — ().
39. De Rosis F., De Carolis B., Pizzutilo S. User tailored hypermedia explanations // INTERCHI'93 Adjunct proc. — Amsterdam,1993. — P. 169–170.
40. Desmarais M.C., Maluf A. User-expertise modeling with empirically derived probabilistic implication networks // User Modeling and User Adapted Interaction. — 1996. — Vol. 5. — P. 283–315.
41. Engelbart D. The augmented knowledge workshop // A History of Personal Workstations. — Addison Wesley, 1988.

42. Faulhaber S. Reinhardt B. D3-WWW-Trainer // Entwiklung einer Oberfläche für die Netzanwendung. — München, Technische Universität München, 1997. P. 31–40.
43. Faulmann C. Illustrierte Geschichte der Schrift. — Augustus Verlag, 1980. Originally published in 1880.
44. Fink J., Kobsa A., Schreck J. Personalized hypermedia information provision through adaptive and adaptable systems features: User modeling, privacy and security issues // Intelligence in Services and Networks: Technology for Cooperative Competition. — Springer-Verlag, 1997. — P. 459–467.
45. Gilbert J. E., Han C.Y. Arthur: Adapting Instruction to Accommodate Learning Style // Proc. of World Conf. of the WWW and Internet (WebNet'99). — Honolulu, HI, 1999. — P. 433–438.
46. Gloor P. Elements of Hypermedia Design. Birkhauser, 1997.
47. Goldstein I. The genetic graph: A represenation for the evolution of procedural knowledge // Intelligent Tutoring Systems. — Academic Press, 1982.
48. Greer J., McCalla G., Collins J., Kumar V., Meagher P., Vassileva J. Supporting peer help and collaboration in distributed workplace environments // Intern. J. of Artificial Intelligence in Education. — 1998. — Vol. 9. — P.159–177.
49. Gronbaek K., Trigg R.H. From Web to Workplace: Designing Open Hypermedia Systems. — The MIT Press, 1999.
50. Halasz F., Schwartz M. The Dexter Hypertext Reference Model // Communs. of the ACM. — 1994. — Vol. 37, N 2. — P. 30–39.
51. Henze N., Nejdl W. Adaptivity in the KBS hyperbook system // Proc. of 2nd Workshop on Adaptive Systems and User Modeling on the WWW. — Toronto, 1999.
52. Henze N., Nejdl W., Wolpers M. Modeling constructivist teaching functionality and structure in the KBS hyperbook system // CSCL'99: Computer Supported Collaborative Learning. — Standford, 1999.
53. Hook K., Karlgren J., Waern A., Dahlback N., Jansson C., Karlgren K., Lemaire B. A glass box approach to adaptive hypermedia // User Modeling and User Adapted Interaction. — 1996. — Vol. 6, N 2-3. — P. 157–184.
54. Hoppe U. Use of multiple student modeling to parametrize group learning // Proc. of 7th World Conference on Artificial Intelligence in Education (AI-ED'95). — Washington, DC, 1995. — P. 234–249.
55. Ito J., Okazaki Y., Watanabe K., Kondo H., Okamoto M.: Pen based user interface for an ITS on WWW client // Proc. of ICCE'98. — Beijing, AACE, 1998. — P. 324–327.
56. Jameson A. Numerical uncertainty management in user and student modeling: An overview of systems and issues // User Modeling and User Adapted Interaction. — 1996. — Vol. 5, N 3-4. — P. 193–251.
57. Jameson A. What can the rest of us learn from research on adaptive hypermedia — and vice versa? . — Saarbrucken, 1999. — (Tech. rep. / University of Saarbrucken).
58. Joerding T. A temporary user modeling approach for adaptive shopping on the Web // Proc. of Second Workshop on Adaptive Systems and User Modeling on the World Wide Web. — Eindhoven University of Technology, 1999. — P. 75–79.

59. Jones K.S., Willet P., Eds. Readings in Information Retrieval. — Morgan Kaufmann, 1997.
60. Kass R. Student modeling in intelligent tutoring systems — implications for user modeling // User Models in Dialog Systems. — Springer, 1989.
61. Kay J., Kummerfeld B. User Models for Customized Hypertext // Lect. Notes Comput. Sci. — 1997. — Vol. 1326.
62. Kay J., Kummerfeld R. An individualised course for the C programming language // Proc. of the 2nd International World Wide Web Conference. — Chicago, 1994.
63. Kinshuk, Han B., Hong H., Ratel A. Stundent adaptivity in TILE // IEEE Intern. Conf. on Advanced Lerning Technologies. — IEEE Computer Society, 2001. — P. 297–300.
64. Kobsa A. User Modeling in Dialog Systems: Potentials and Hazards. // AI & Society. — 1990. — Vol. 4. — P. 214—240.
65. Kobsa A. User Modeling: Recent Work, Prospects and Hazards. // Adaptive User Interfaces: Principles and Practice. — Amsterdam, North Holland Elsevier. — 1993.
66. Kobsa A., Pohl W. The user modeling shell system BGP-MS. — Konstanz, 1995 — (Tech. rep. / University of Konstanz).
67. López J.M., Millán E., Pérez-de-la-Cruz J.L., Triguero F. ILESA: a Web-based intelligent learning environment for the simplex algorithm // Proc. of CALISCE'98, 4th Intern. Conf. on Computer Aided Learning and Instruction in Science and Engineering. — Göteborg, 1998. — P. 399–406.
68. Lowe D., Hall W. Hypermedia and the Web. — J. Wiley and Sons, 1999.
69. Martin J., Van Lehn, J. OLAE: Progress toward a multi-activity, Bayesian student modeler // Proc. of Artificial intelligence in education AIED. — Edinburgh, 1993. — P. 441–417.
70. Mislevy R., Gitomer D. The role of probability-based inference in an intelligent tutoring system // User Modeling and User-Adapted Interaction. — 1996. — Vol. 5. — P. 253–282.
71. Nelson T. Replacing the printed word: A complete literary system // Proc. of IFIP Congress. — Netherlands, 1980. — P. 1013–1023.
72. Nielsen J. Multimedia, Hypertext und Internet: Grundlagen und Praxis des elektronischen Publizierens. — Vieweg, 1995.
73. Nissen H., Jeusfeld M., Jarke M., Zemanek G., and Huber, H. Requirements analysis from multiple perspectives: Experiences with conceptual modeling technology // IEEE Software. — 1996. — Vol. 13, N 2.
74. Nyce J., and Kahn P. A Machine for the Mind: Vannevar Bush's Memex. // From Memex to Hypertext: Vannevar Bush and the Mind's Machine. — Academic Press, 1991.
75. Okazaki Y., Watanabe K., Kondo H. An Implementation of an intelligent tutoring system (ITS) on the World-Wide Web (WWW) // Educational Technology Research. — 1996. — Vol. 19, N 1. — P. 35–44.
76. Paiva A., Machado I. Vincent, an autonomous pedagogical agent for on-the-job training // Lect. Notes Comput. Sci. —1998. — Vol. 1452. — P. 584–593.

77. Pearl J. Probabilistic Reasoning // Intelligent Systems: Networks of Plausible Inference. — Morgen Kaufmann Publishers, 1988.
78. Pérez T., Gutiérrez J., Lopistéguy P. An adaptive hypermedia system // Proc. of 7th World Conf. on Artificial Intelligence in Education (AI-ED'95). — Washington, DC, 1995. — P. 351–358.
79. Prentzas J., Hatzilygeroudis I., Koutsojannis C. A Web-based ITS controlled by a hybrid expert system // Proc. of IEEE Intern. Conf. on Advanced Lerning Technologies. — IEEE Computer Society, 2001. — P. 131–134.
80. Rada R. Interactive Media. — Springer, 1995.
81. Rich E. User modeling via stereotypes // Cognitive Science. — 1978. —N 3. — P. 329–354.
82. Ritter S. PAT-Online: A Model-tracing tutor on the World-wide Web // Proc. of Workshop «Intelligent Educational Systems on the World Wide Web» at AI-ED'97, 8th World Conf. on Artificial Intelligence in Education — Kobe, ISIR, 1997. — P. 11—17.
83. Rosis F.D., Pizzutilo S., Russo A., Berry D.C., Molina F. J. Modeling the user knowledge by belief networks // User Modeling and User Adapted Interaction. — 1992. — Vol. 2. — P. 367–388.
84. Schafer R., Weyrath T. Assessing temporally variable user properties with dynamic Bayesian networks // Proc. of the Sixth Internat. Conf. on User Modeling (UM97). — Sardinia, 1997.
85. Schank R., Cleary C. Engines for Education. — Lawrence Erlbaum Associates, 1994.
86. Specht M. Empirical evaluation of adaptive annotation in hypermedia // ED-Media and ED-Telekom. — Freiburg, 1998.
87. Specht M., Oppermann R. ACE — Adaptive Courseware Environment // The New Review of Hypermedia and Multimedia. — 1998. — Vol. 4. — P. 141–161.
88. Taylor J.C. Fifth Generations Distance Education // Proc. of 20th ICDE World Conf. on Open learning and Distance Education. — Dusseldroff, 2001.
89. Vassileva J. A task-centered approach for user modeling in a hypermedia office documentation system // User Modeling and User-Adapted Interaction. — 1996. — Vol 6, N 2–3.
90. Virvou M., Tsiriga V. Web-passive voice tutor: an intelligent computer assisted language learning system over the WWW // Proc. of IEEE Intern. Conf. on Advanced Lerning Technologies. — IEEE Computer Society, 2001. — P. 131–134.
91. Wang T., Hornung C. The Modular Training System (MTS) — a system architecture for Internet-based learning and training. — Fraunhofer, Fr-IGD, 1997.
92. Warendorf K., Tan C. ADIS — An animated data structure intelligent tutoring system or Putting an interactive tutor on the WWW // Proc. of Workshop «Intelligent Educational Systems on the World Wide Web» at AI-ED'97, 8th World Conf. on Artificial Intelligence in Education. — Kobe, ISIR, 1997. — P. 54–60.
93. Weber G. Episodic learner modeling // Cognitive Science. —1996. — Vol. 20.

94. Weber G., Mollenberg A. ELM programming environment: A tutoring system for lisp beginners // Cognition and Computer Programming. — Ablex Publishing Corporation, 1995.
95. Weber G., Specht M. User modeling and adaptive navigation support in WWW-based tutoring systems // Proc. of the Sixth Intern. Conf. on User Modeling (UM97). — Sardinia, 1997.
96. Wu H., De Bra P., Aerts A., Houben G.-J. Adaptation control in adaptive hypermedia systems // Lect. Notes Comput. Sci. — 2000. — Vol.1892. — P. 250–259.
97. Wu H., De Bra P., Aerts A., Houben G.J., Adaptation Control in Adaptive Hypermedia Systems. // Lect. Notes. Comput. Sci. — 2000. — Vol. 1892. — P. 250–259.
98. Wu H., De Kort E., De Bra P. Design Issues for General-Purpose Adaptive Hypermedia Systems. // Proc. of the ACM Conf. on Hypertext and Hypermedia. — Aarhus, 2001. — P. 141–150.
99. Wu H., De Kort E., De Bra P. Dessign issues for general-purpose adaptive hypermedia systems // Proc. of the 12th ACM Conf. On Hypertext and Hypermedia. — Aarhus, 2001. — P.141–150.
100. Wu H., Houben G.J., De Bra P. Supporting User Adaptation in Adaptive Hypermedia Applications. // On-line Conf. and Informatiewetenschap 2000. — De Doelen, Rotterdam, 2000.

В. Н. Касьянов, И. Л. Мирзуитова

РЕСТРУКТУРИРУЮЩИЕ ПРЕОБРАЗОВАНИЯ: АЛГОРИТМЫ РАСПАРАЛЛЕЛИВАНИЯ ЦИКЛОВ*

1. ВВЕДЕНИЕ

Существует три основных подхода к обеспечению эффективной эксплуатации суперкомпьютеров как машин с параллельной архитектурой: использование параллельных языков, использование библиотек и автоматическое распараллеливание программ. Все три направления интенсивно развиваются, но наиболее привлекательным остается третий подход, поскольку решает проблему переносимости старых последовательных программ на суперкомпьютеры.

Несмотря на то что автоматическое распараллеливание программ — достаточно трудная задача, существующее ее решение обеспечило коммерческий успех суперкомпьютеров на быстро меняющемся рынке вычислительной техники. В настоящее время по грубой оценке в мире эксплуатируется порядка тысячи суперкомпьютеров различных параллельных архитектур с производительностью, измеряемой в гигафлопах и даже выше. На большинстве из них в качестве языка программирования используется Фортран (Фортран 77, Фортран 90), а в состав программного обеспечения входит распараллеливающий Фортран-компилятор.

Создание успешно работающих распараллеливающих компиляторов для ЭВМ с параллельной архитектурой (векторно-конвейерные ЭВМ, мультипроцессорные системы, машины с VLIW-архитектурой) стало возможным за счет развития методов оптимизирующей трансляции. Произошло существенное расширение класса преобразований, необходимых для достижения приемлемой эффективности при трансляции с языков высокого уровня, за счет включения в него конвейеризующих, векторизующих и других преобразований, ориентированных на те или иные особенности архитектур перспективных ЭВМ [1–5, 7, 8, 10–12, 23, 18–22, 30, 31]. Класс оптимизаций

* Работа выполнена при финансовой поддержке Научной программы «Университеты России» (грант № УР.04.01.027) и Министерства образования РФ (грант № Е02-1.0-42).

расширился не только преобразованиями программ в рамках одной модели вычислений, например некоторой модели последовательных или асинхронных вычислений, но и так называемыми *конструирующими* преобразованиями[1], переводящими программы из одних моделей в другие, например, сопоставляющими определенным последовательным программам эквивалентные им параллельные. При этом процессы оптимизирующей трансляции, так или иначе реализуемые распараллеливающими компиляторами, осуществляются по «типовой» схеме трансформационного конструирования эффективной программы (трансформационного синтеза) [15, 16]. Схема предполагает последовательное выполнение следующих шагов.

1. Применение оптимизирующих преобразований, позволяющих приводить программу в рамках исходной модели вычислений к ориентированному на применение конструирующих преобразований виду.
2. Применение конструирующих преобразований, переводящих программу из исходной модели в результирующую.
3. Применение оптимизирующих преобразований в рамках результирующей модели.

Расширение оптимизаций привело к следующим изменениям в их классификации [10].

Поскольку параллельные машины обладают большим разнообразием архитектур, наряду с разбиением, имеющим место для оптимизаций для последовательных ЭВМ, произошло выделение в классе машинно-зависимых оптимизаций не только машинно-ориентированных, но и архитектурно-ориентированных оптимизаций. Этот подкласс, как и машинно-ориентированные преобразования, содержит оптимизации, в которых зависимости от конкретной машины могут быть вынесены в набор параметров, что позволяет реализовать их не на уровне машинного языка конкретной ЭВМ. Фактически машинно-ориентированные оптимизации можно рассматривать как архитектурно-ориентированные преобразования для класса последовательных машин. Степень зависимости преобразований от конкретной архитектуры может быть различной, например, можно различать мелкозернистые оптимизации, которые ориентированы на архитектуры, эксплуатирующие мелкозернистый параллелизм, и VLIW-оптимизации, реализуемые в рамках языка абстрактной VLIW-машины [31].

В рамках разделения оптимизаций по способу реализации (смешанная стратегия, оптимизирующие преобразования) произошли следующие изме-

[1] Здесь мы используем классификацию оптимизаций, предложенную В.Н. Касьяновым; она является обобщением предложенной им классификации оптимизаций для последовательных компьютеров [15] и подробно описана в [9].

нения. Смешанная стратегия стала частью подкласса конструирующих преобразований, который наряду с преобразованиями для последовательных программ, такими как, например, заменяющие рекурсию на цикл, включает так называемые *распараллеливающие* преобразования, связанные с построением параллельной программы по последовательной: векторизующие преобразования [3]; преобразования компактификации или упаковки [1, 20, 31] и т. п. Что касается класса оптимизирующих преобразований, то он расширился за счет оптимизаций параллельных программ и так называемых *реструктурирующих* преобразований, которые ориентированы на приведение преобразуемой программы к виду, более подходящему для выполнения конструирующих преобразований. Следует отметить, что последнее разделение не является абсолютным: одно и то же преобразование в одном трансляторе (в рамках одной системы преобразований) может быть оптимизирующим, а в другом — реструктурирующим.

Основная задача распараллеливающего компилятора — извлечь как можно больше скрытого параллеллизма из участков повторяемости последовательной программы, определяющих основное время ее выполнения: циклов и рекурсивных процедур. Это требует изменения порядка исполнения операторов в участках повторяемости последовательной программы, которые до выполнения алгоритмов распараллеливания, в случае отсутствия рекурсивных процедур, можно привести к виду гнезд DO-циклов (см., например, [5]). Поэтому гнездо DO-циклов берется в качестве модели программы для рассматриваемых в статье алгоритмов распараллеливания. Условия возможности проведения тех или иных преобразований выражаются в виде так называемых программных зависимостей. Зависимость в общем смысле есть отношение между двумя вычислениями, которое налагает ограничения на порядок их исполнения. Статический анализ зависимостей идентифицирует эти ограничения и дает информацию для проведения реструктурирующих преобразований во время трансляции.

Алгоритмы распараллеливания циклов — это весьма важный класс реструктурирующих преобразований. Они особенно привлекательны тем, что их применение не требует знания целевой архитектуры. Эти алгоритмы можно рассматривать как завершающую стадию машинно-независимой части процесса генерации параллельного кода распараллеливающим компилятором. Распараллеливание циклов позволяет обнаружить параллелизм (преобразуя циклы DO в циклы DOALL) и выявить те зависимости, которые ответственны за изначально «последовательное» состояние некоторых операций исходной программы.

Разумеется, следующая стадия генерации кода, связанная с выполнением конструирующих преобразований и преобразований параллельных программ, должна будет принимать во внимание параметры целевой архитектуры. Нахождение подходящего уровня зернистости является ключом к высокой производительности. Кроме того, необходимо рассматривать такие важные аспекты, как распределение данных и оптимизацию коммуникаций. Все эти проблемы весьма важны, но в данной работе не будут рассматриваться.

Данная работа посвящена изучению различных алгоритмов обнаружения параллелизма (в гнездах циклов), основанных на:

1) простой декомпозиции графа зависимостей на сильно связные компоненты (такие как алгоритм Аллена и Кеннеди [24]);

2) унимодулярных преобразованиях цикла — либо специальных преобразованиях (алгоритм Банержи [27]), либо автоматически генерируемых (алгоритм Вольфа и Лэма [60]);

3) планировании — либо одномерном [37, 39, 46] (особый случай — метод гиперплоскостей [55]), либо многомерном [42, 47].

Существующие алгоритмы распараллеливания циклов различаются по многим направлениям. Во-первых, они используют различные математические инструменты: графовые алгоритмы в случае (1), матричные вычисления в (2), линейное программирование в (3). Во-вторых, они принимают на входе различные описания зависимостей по данным: графовое описание и уровни зависимостей в (1), векторы направления в (2), описание зависимостей в виде многогранников или аффинных выражений в (3). Для каждого из этих алгоритмов мы определим основные понятия, лежащие в их основе, и обсудим их сильные и слабые стороны как на примерах, так и в сравнении «оптимальных» результатов.

Нас будет интересовать характеристика того, какой алгоритм наилучшим образом подходит для того или иного заданного представления зависимостей. Понятно, что нет необходимости использовать усложненный алгоритм анализа зависимостей, если алгоритм распараллеливания не сможет воспользоваться точностью его результата. И наоборот, нет смысла в использовании точнейшего алгоритма распараллеливания, если представление зависимостей не будет достаточно точным.

Оставшаяся часть работы организована следующим образом. Разд. 2 посвящен краткому обзору предмета рассмотрения, а именно — алгоритмов распараллеливания циклов. В разд. 3 мы рассмотрим основные абстракции зависимостей: уровни зависимостей, векторы направлений, многогранники зависимостей. Алгоритм Аллена—Кеннеди [24] приведен в разд. 4, алго-

ритм Вольфа—Лэма [60] — в разд. 5. Показано, что оба алгоритма являются «оптимальными» в классе алгоритмов распараллеливания, использующих ту же самую абстрактную конструкцию в качестве входного представления, а именно — уровни зависимостей у Аллена и Кеннеди и векторы направлений у Вольфа и Лэма. В разделе 6 мы представим алгоритм Дартье—Вивьена [36], охватывающий оба предыдущих. Он основан на обобщении векторов направления — многогранниках зависимостей. В разд. 7 мы приведем краткий обзор алгоритма Фотрие [46, 47], основанного на точных аффинных зависимостях. Наконец, в разд. 8 будут высказаны некоторые заключения.

2. ВХОД И ВЫХОД РАСПАРАЛЛЕЛИВАЮЩИХ АЛГОРИТМОВ

Вложенные DO-циклы в состоянии описывать объем вычислений, размер которого значительно превосходит размер соответствующей программы. Например, рассмотрим n вложенных циклов, счетчики которых описывают n-мерный куб с ребром N: эти циклы вмещают в себя объем вычислений размером N^n. Часто случается, что такие гнезда циклов содержат ненулевую *степень параллелизма*, т. е. множество независимых вычислений размером $\Omega(N^r)$ для $r \le 1$.

Это делает задачу распараллеливания вложенных циклов очень трудной и интересной: распараллеливающий компилятор должен уметь определять по возможности ненулевую степень параллелизма за время, *непропорциональное* по сравнению с последовательным исполнением цикла. Чтобы такое стало возможным, эффективные алгоритмы распараллеливания должны иметь *сложность*, *входной размер* и *выходной размер*, зависящие только от n, но ни в коем случае не от N, т. е. зависящие от размера последовательного кода, а не от количества описываемых им вычислений.

Входом распараллеливающего алгоритма является описание зависимостей, которые связывают отдельные вычисления. Выходом является описание эквивалентного кода с явным параллелизмом.

2.1. Вход: граф зависимостей

Каждый оператор гнезда циклов окружен несколькими циклами. Каждая итерация этих циклов определяет конкретное исполнение оператора, называемое *операцией*. Зависимости между операциями представлены ориенти-

рованным ациклическим графом[2] — *расширенным графом зависимостей* (РГЗ). Количество вершин РГЗ равно количеству операций в гнезде циклов. Исполнение операций гнезда циклов в соответствии с определяемым РГЗ частичным порядком гарантирует сохранение корректности результата вычисления. Распознавание параллелизма в гнезде циклов сводится к нахождению цепей антизависимостей в РГЗ. Понятие «расширенного графа зависимостей» иллюстрирует пример 1. Соответствующий РГЗ изображен на рис. 1.

Пример 1.

```
DO i=1,n
    DO j=1,n
        a(i,j)=a(i-1,j-1)+a(i,j-1)
    ENDDO
ENDDO
```

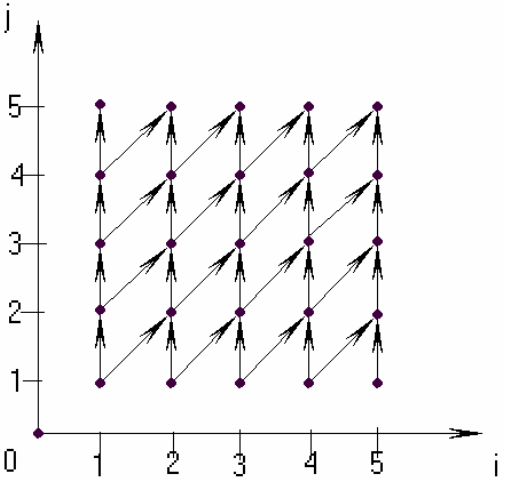


Рис. 1. РГЗ фрагмента примера 1

[2] Здесь и далее мы используем стандартную терминологию теории графов (см., например, [13]).

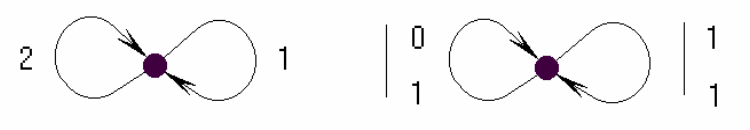
К сожалению, РГЗ не может быть использован в качестве входного представления для распараллеливающих алгоритмов, так как обычно он слишком велик и не может быть точно описан во время компиляции. Вместо него используется так называемый *сокращенный граф зависимостей* (*СГЗ*) — сжатое и приближенное представление РГЗ. Это приближение должно быть надмножеством РГЗ, чтобы сохранить отношения зависимости. В СГЗ имеется одна вершина для каждого оператора гнезда циклов, его дуги помечены в соответствии с выбранным приближением зависимостей (см. более подробное изложение в разделе 3). На рис. 2 представлены два возможных СГЗ для программы из примера 1, соответствующих двум различным приближениям зависимостей.

Поскольку входом является СГЗ, а не РГЗ, распараллеливающий алгоритм не способен отличить два различных РГЗ, имеющих один СГЗ. Следовательно, удастся распознать только параллелизм, содержащийся в СГЗ. Таким образом, качество распараллеливающего алгоритма должно оцениваться относительно анализа зависимостей.

Например, программы из примеров 1 и 2 имеют один и тот же СГЗ с уровнями зависимостей (рис. 2(а)). Таким образом, распараллеливающий алгоритм, принимающий на входе СГЗ с уровнями зависимостей, не в состоянии различить две программы. Однако программа из примера 1 содержит один уровень параллелизма, тогда как программа из примера 2 является существенно последовательной.

Пример 2.

```
DO i = 1, n
    DO j = 1, n
        a(i, j) = 1 + a(i-1, n) + a(i, j-1)
    ENDDO
ENDDO
```



а *б*

Рис. 2. Сокращенный граф зависимостей: (а) с уровнями зависимостей; (б) с векторами направлений

2.2. Выход: вложенные циклы

Размер распараллеленного кода, как упоминалось ранее, не должен зависеть от количества описываемых им операций. В этом причина того, что результат работы распараллеливающего алгоритма всегда должен описываться множеством циклов[3].

Имеются по крайней мере три способа определить новый порядок на операциях заданного гнезда циклов (то есть три способа определить выход распараллеливающего алгоритма) в терминах вложенных циклов.

1. Использование элементарных преобразований циклов в качестве базовых шагов алгоритма, таких как распределение цикла (в алгоритме Аллена и Кеннеди), перестановка или перекос циклов (как в алгоритме Банержи).
2. Линейное изменение базы области итераций, то есть применение унимодулярного преобразования на векторах итераций (как в алгоритме Вольфа и Лэма).
3. Определение d-мерного плана, т. е. применение аффинного преобразования из Z^n в Z^d и интерпретация преобразования как многомерной временной функции. Каждая компонента соответствует последовательному циклу, а недостающие (n – d) размерностей соответствуют циклам DOALL (как в алгоритмах Фотрие и Дарте—Вивьен).

Результат работы этих трех схем преобразований, разумеется, может быть описан в терминах гнезд циклов после более или менее сложного процесса переписывания (см. [32, 35, 39, 60, 65]). Здесь мы не будем говорить о переписывании. Вместо этого мы сосредоточимся на связи между представлением зависимостей (входом) и преобразованиями циклов, используемыми распараллеливающим алгоритмом (выходом). Нашей целью является формулировка того, какой алгоритм окажется оптимальным для заданного представления зависимостей. «Оптимальный» означает здесь, что алгоритм в состоянии выявить максимальное количество параллельных циклов.

[3] Эти циклы могут быть произвольно сложными, при условии, что их сложность зависит только от размера исходного кода. Очевидно, что чем проще результат, тем лучше. Но в этом контексте значение слова «проще» не совсем тривиально, поскольку все зависит от оптимизаций, которые могут последовать далее. Обычно полагается, что структурная простота предпочтительна.

3. АБСТРАКЦИИ ЗАВИСИМОСТЕЙ

Для ясности мы ограничим изложение случаем совершенного гнезда DO-циклов с аффинными границами. Это ограничение позволяет описать итерации n вложенных циклов (n называется *глубиной* гнезда циклов) с векторами из Z^n (называемыми *векторами итераций*), содержащимися в конечном выпуклом многограннике (называемом *областью итераций*), ограниченном границами циклов. i-я компонента вектора итерации является значением счетчика i-го цикла в гнезде, считая от самого внешнего к самому внутреннему. Таким образом, в последовательной программе итерации исполняются в лексикографическом порядке их векторов итераций.

Введем следующие обозначения:

D — область итераций в виде многогранника; **I** и **J** — n-мерные векторы итераций в **D**; S_i — i-й оператор в гнезде циклов, где $1 \le i \le s$. Мы будем писать $\mathbf{I} >_l \mathbf{J}$, если **I** лексикографически больше **J**, и $\mathbf{I} \ge_l \mathbf{J}$, если $\mathbf{I} >_l \mathbf{J}$ или $\mathbf{I} = \mathbf{J}$.

В разд. 3.1 описываются различные концепции графов зависимостей, неформально введенные в разд. 2.1: расширенные графы зависимостей (РГЗ), сокращенные графы зависимостей (СГЗ), графы истинных зависимостей (ГИЗ), а также понятие множеств расстояний. В разд. 3.2 мы формально определим понятие многогранного сокращенного графа зависимостей (МСГЗ), то есть сокращенного графа зависимостей, дуги которого помечены многогранниками. Наконец в разд. 3.3 мы покажем, как модель МСГЗ обобщает классические абстракции множеств расстояний, такие как уровни зависимостей и векторы направления.

3.1. Графы зависимостей и множества расстояний

Отношения зависимостей между операциями определяются условиями Бернштейна [30]. Вкратце, две операции связаны зависимостью, если обе они обращаются к одной и той же области памяти и по меньшей мере одно из обращений производит запись в эту область. Зависимость имеет направление в соответствии с последовательным порядком, от первой исполненной операции к последней. В зависимости от порядка операций записи и/или чтения, зависимость называется *потоковой зависимостью, антизависимостью* или *выходной зависимостью*. Мы будем писать $S_i(I) \Rightarrow S_j(I)$, если оператор S_j на итерации J зависит от оператора S_i на итерации I. Частичный порядок, определяемый отношением $\Rightarrow$, описывает расширенный граф за-

висимостей (РГЗ). Заметим, что (J - I) всегда является лексикографически ненулевым, когда $S_i(I) \Rightarrow S_j(J)$.

В общем случае РГЗ не может быть вычислен во время компиляции, либо в силу недостатка информации (такой как значения параметров границ цикла или даже точное число обращений к памяти), либо из-за того, что генерация полного графа обходится слишком дорого (см. в [61, 66] обзор алгоритмов проверки зависимостей, таких как проверка наибольших общих делителей [24, 28], тест Банержи [27–29], ω-тест [58], λ-тест [56], I-тест [48]; в [45] — более детальное изложение точного анализа зависимостей). Вместо этого рассматривается представление зависимостей в виде меньшего по размеру ориентированного графа с циклами, имеющего s вершин (по количеству операторов), который называется *сокращенным графом зависимостей* (*СГЗ*) (или *графом зависимостей на уровне операторов*).

СГЗ представляет собой сжатие РГЗ. В СГЗ два оператора S_i и S_j являются зависимыми (мы будем обозначать это как e: $S_i \to S_j$), если существует по меньшей мере одна пара (I, J) такая, что $S_i(I) \Rightarrow S_j(J)$. Дуга[4] e от S_i к S_j в СГЗ помечена множеством $\{(I, J) \in D^2 \mid S_i(I) \Rightarrow S_j(J)\}$ или аппроксимацией D_e, содержащей это множество. Точность этой аппроксимации и ее представление и составляют силу алгоритма анализа зависимостей.

Другими словами, СГЗ описывает в сокращенной форме граф зависимостей на уровне итераций, называемый (*максимальным*) *графом истинных зависимостей* (*ГИЗ*), представляющий собой подмножество РГЗ. ГИЗ и РГЗ имеют одни и те же вершины, но ГИЗ имеет большее количество дуг, определяемых следующим образом:

$$(S_i,I) \Rightarrow (S_j,J) \text{ (в ГИЗ)} \Leftrightarrow$$
$$\exists\, e = (S_i, S_j) \text{ (в РГЗ), такая, что } (I, J) \in D_e.$$

Для определенного класса вложенных циклов существует возможность выразить в точности это множество пар (I, J) (см. [45]): I задается как аффинная функция (в некоторых конкретных случаях, включающих функции нахождения ближайшего целого снизу или сверху) $f_{i,j}$ от J, где J изменяется в пределах многогранника $P_{i,j}$:

$$\{(I, J) \in D^2 \mid S_i(I) \Rightarrow S_j(J)\} = \{(f_{i,j}(J),J) \mid J \in P_{i,j} \subset D\}. \qquad (1)$$

[4] Такая дуга существует для каждой пары обращений к памяти, вызывающих зависимость между S_i и S_j.

Однако большинство алгоритмов анализа зависимостей вместо множества пар (I, J) вычисляет множество $E_{i,j}$ всех возможных значений (J - I). $E_{i,j}$ называется множеством *векторов расстояния*, или *множеством расстояний*:

$$E_{i,j} = \{(J - I) \mid S_i(I) \Rightarrow S_j(J)\}.$$

В случаях, когда возможно провести точный анализ зависимостей, равенство 1 показывает, что множество векторов расстояния является проекцией целочисленных точек многогранника. Это множество может быть аппроксимировано его выпуклой оболочкой или более или менее точным многогранником большего размера (или конечным объединением многогранников). Когда множество векторов расстояния представлено конечным объединением многогранников, соответствующая дуга зависимостей в СГЗ разлагается на кратные дуги.

Заметим, что представление с помощью векторов расстояния не эквивалентно представлению с помощью пар (как в равенстве 1), поскольку теряется информация о *расположении* в РГЗ этого вектора. Это может вызвать даже некоторое уменьшение количества параллелизма, как будет показано в примере 9. Однако такое представление все же остается полезным, особенно в случаях, когда точный анализ зависимостей слишком накладен или вообще неосуществим.

Классическими представлениями множеств расстояний (в порядке уменьшения точности) являются:

- *уровни зависимостей*, введенные и использованные в распараллеливающем алгоритме Аллена и Кеннеди [24, 25];
- *векторы направления*, введенные Лампортом [55] и Вольфом [62, 63], а затем использованные в распараллеливающем алгоритме Вольфа и Лэма [60];
- *многогранники зависимостей*, введенные в работе [50] и использованные в алгоритме с разделением супервершин Иригойна и Триоле [51]. Более подробное описание многогранников зависимостей можно найти в работах по проекту PIPS [49].

Теперь мы дадим формальное определение сокращенных графов зависимостей, дуги которых помечены многогранниками зависимостей. Затем мы покажем, что это представление объединяет два других представления, а именно — уровни зависимостей и векторы направления.

3.2. Многогранные сокращенные графы зависимостей

Вспомним математическое определение многогранника и то, как можно провести его декомпозицию на вершины, лучи и линии.

Множество P векторов в пространстве Q^n называется (*выпуклым*) *многогранником,* если существуют целочисленная матрица A и целочисленный вектор b такие, что

$$P = \{x \mid x \in Q^n, Ax \le b\}.$$

Политопом называется ограниченный многогранник.

Справедливы следующие свойства. Любой многогранник может быть разложен на сумму политопа и многогранного конуса. Политоп определяется его вершинами, и каждая точка политопа является неотрицательной барицентрической комбинацией вершин политопа. Многогранный конус может быть определен с помощью его лучей и линий. Любая точка многогранного конуса есть сумма неотрицательной комбинации его лучей и произвольной комбинации его линий.

Таким образом, многогранник зависимостей P может быть эквивалентно определен с помощью множества *вершин* (обозначаемых $\{v_1, ..., v_\omega\}$), множества *лучей* (обозначаемых $\{r_1, ..., r_\rho\}$) и множества *линий* (обозначаемых $\{l_1, ..., l_\lambda\}$). Тогда P есть множество всех векторов p таких, что

$$p = \sum_{i=1}^{w} \mu_i \upsilon_i \quad + \quad \sum_{i=1}^{\rho} \upsilon_i r_i \quad + \quad \sum_{i=1}^{\lambda} \xi_i l_i \ , \tag{2}$$

где $\mu_i \in Q^+$, $\nu_i \in Q^+$, $\xi_i \in Q$ и $\sum_{i=1}^{w} \mu_i = 1$.

Теперь мы определим то, что называется многогранным сокращенным графом зависимостей (или МСГЗ), а именно — сокращенный граф зависимостей, помеченный многогранником зависимостей. На самом деле нам нужны только целочисленные векторы, принадлежащие к многограннику зависимостей, поскольку расстояния зависимостей в сущности являются целочисленными векторами.

Многогранный сокращенный граф зависимостей (МСГЗ) есть СГЗ, в котором каждая дуга e: $S_i \to S_j$ помечена многогранником P(e), аппроксимирующим множество векторов расстояний: ассоциированный истинный граф зависимостей содержит дугу из экземпляра I вершины S_i к экземпляру J вершины S_j в том и только том случае, когда $(J - I) \in P(e)$.

В разд. 6 это представление зависимостей рассматривается более подробно; пока будем воспринимать многогранник зависимостей как некоторое обобщение векторов направления.

3.3. Векторы направлений

Когда множество векторов расстояния содержит только один элемент, зависимость называется униформной, и единственный вектор расстояния называется *униформным вектором зависимости.*

В противном случае множество векторов расстояния может быть представлено n-мерным вектором (*вектором направления*), компоненты которого принадлежат к $Z \cup \{*\} \cup (Z \times \{+, -\})$. Его i-я компонента представляет собой аппроксимацию i-х компонент всех возможных векторов расстояния: она равна z+ (соответственно z-), если i-я компонента больше (соответственно меньше) или равна z. Она равна *, если i-я компонента может принимать любое значение, и z, если зависимость является униформной по этой размерности с единственным значением z. Обычно + (соответственно -) используется как сокращение для 1+ (соответственно для (-1)-).

Мы обозначаем как e_i i-й канонический вектор, то есть n-мерный вектор, все компоненты которого являются нулевыми, за исключением i-й компоненты, равной 1. Тогда вектор направления — не что иное, как аппроксимация многогранником, имеющим одну вершину, все лучи и линии которого, если они имеются, суть канонические векторы.

Действительно, рассмотрим дугу e, помеченную вектором направления d, и обозначим через I^+, I^- и I^* множества компонент d, которые равны z+ (для некоторого целого z), z- и *, соответственно. Наконец, обозначим через d_z n-мерный вектор, чья i-я компонента равна z, если i-я компонента d равна z, z+ или z-, и равна 0 в противном случае.

Теперь, в силу определения символов +, - и *, вектор направления d представляет в точности все n-мерные векторы p, для которых существуют такие целые (v, v', ξ) в $N^{|I+|} \times N^{|I-|} \times Z^{|I*|}$, что

$$p = d_z + \sum_{i \in I^+} v_i e_i - \sum_{i \in I^-} v_i' e_i + \sum_{i \in I^*} \xi_i e_i \ . \qquad (3)$$

Другими словами, вектор направления d представляет все целые точки, которые принадлежат многограннику, определяемому единственной вершиной d_z, лучами e_i для $i \in I^+$, лучами $-e_i$ для $i \in I^-$ и линиями e_i для $i \in I^*$.

Например: вектор направления (2+, *, -, 3) определяет многогранник с одной вершиной (2, 0, -1, 3), двумя лучами (1, 0, 0, 0) и (0, 0, -1, 0) и одной линией (0, 1, 0, 0).

3.4. Уровни зависимостей

Представление с помощью уровня является менее точной абстракцией зависимостей. В гнезде из n вложенных циклов множество векторов расстояния аппроксимируется целым числом l из интервала $[1, n] \cup \{\infty\}$, которое определяется как наибольшее целое, такое, что первые $l - 1$ компонент векторов расстояний оказываются нулевыми.

Зависимость на уровне $l \le n$ означает, что зависимость обнаруживается на уровне l гнезда циклов, т. е. на заданной итерации $l - 1$ внешних циклов. В этом случае говорят, что зависимость является *циклически порождаемой зависимостью* на уровне l. Если $l = \infty$, то зависимость происходит внутри тела цикла, между двумя различными операторами, и называется *циклически независимой зависимостью*. Сокращенный граф зависимостей, дуги которого помечены уровнями зависимостей, называется *сокращенным уровневым графом зависимостей* (*СУГЗ*).

Рассмотрим дугу e уровня l. Согласно определению уровня, первой ненулевой компонентой вектора расстояния является l-я компонента, и теоретически она может принимать любое положительное целое значение. Об оставшихся компонентах у нас нет никакой информации. Следовательно, дуга уровня $l < \infty$ эквивалентна вектору направления (0, … , 0, 1+, ∗, … ,∗), начинающемуся с (l-1) нулевой компоненты, а дуга уровня ∞ соответствует нулевому вектору зависимости. Подобно тому как любой вектор направления допускает построение эквивалентного многогранника, то же можно проделать и с представлением с помощью уровня. Например, зависимость уровня 2 в трехмерном гнезде циклов дает вектор направления (0, 1+, ∗), который соответствует многограннику с одной вершиной (0, 1, 0), одним лучом (0, 1, 0) и одной линией (0, 0, 1).

4. АЛГОРИТМ АЛЛЕНА—КЕННЕДИ

Алгоритм Аллена—Кеннеди [24] был первоначально разработан для векторизации циклов. Затем он был расширен с тем, чтобы максимизировать количество параллельных циклов и минимизировать количество синхронизаций в преобразованном коде. Алгоритм принимает на входе сокращенный уровневый граф зависимостей.

Алгоритм базируется на следующих фактах.

1. Цикл является параллельным, если он не содержит циклически порождаемых зависимостей, т.е. если не существует зависимости, уро-

вень которой равен глубине цикла, которая относится к оператору, окруженному циклом.

2. Все итерации оператора S_1 могут быть осуществлены перед любой итерацией оператора S_2, если в СУГЗ не существует зависимости из S_2 в S_1.

Свойство (1) позволяет пометить цикл как DOALL или DOSEQ, тогда как свойство (2) указывает на то, что нахождение параллелизма может быть проведено независимо в каждой сильно связной компоненте СУГЗ. Извлечение параллелизма производится посредством разделения циклов.

4.1. Алгоритм

Для графа зависимостей G через G(k) обозначается такой его подграф, который получается из G удалением всех зависимостей на уровнях, строго меньших k. Ниже приведен набросок алгоритма Аллена—Кеннеди в его базовой формулировке. Работа алгоритма начинается с вызова ALLEN-KENNEDY(СУГЗ, 1).

```
проц ALLEN-KENNEDY(G: граф, k: целое)=
    если k≤n то
        Произвести декомпозицию G(k) на сильно связные компоненты
      Gi и топологически отсортировать их;
        Переписать код таким образом, чтобы каждая Gi принадлежала
      своему гнезду циклов (на уровне k) и сохранялся порядок на Gi
      (распределение циклов на уровне больше или равном k);
        для всех Gi цикл
                если Gi не имеет дуг на уровне k то
                        пометить цикл на уровне k как цикл DOALL
                иначе пометить его как цикл DOSEQ
                все
        все;
        для всех Gi цикл ALLEN-KENNEDY(Gi, k+1) все
    все
все.
```

Проиллюстрируем работу алгоритма на следующем фрагменте кода:

Пример 3.

```
DO i = 1, n
DO j = 1, n
      DO k = 1, n
      S1: a(i, j, k) = a(i-1, j+i, k) + a(i, j, k-1) + b(i, j-1, k)
      S2: b(i, j, k) = b(i, j-1, k+j) + a(i-1, j, k)
  ENDDO
ENDDO
ENDDO
```

Граф зависимостей G = G(1) для данного гнезда циклов приведен на рис. 3. Он имеет только одну сильно связную компоненту и по меньшей мере одну дугу на уровне 1, в силу чего первый вызов процедуры обнаружит, что самый внешний цикл является последовательным. Тем не менее, на уровне 2 (дуга на уровне 1 более не рассматривается) G(2) имеет две сильно связных компоненты: все итерации оператора S_2 могут быть вынесены перед итерациями оператора S_1.

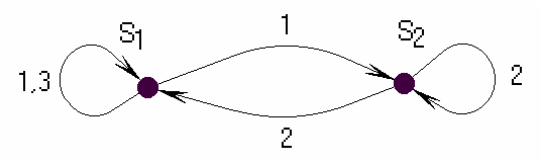


Рис. 3. СУГЗ для программы примера 3

Проведем разделение цикла. Сильно связная компонента, включающая S_1, не содержит дуг на уровне 2 и содержит одну дугу на уровне 3. Таким образом, второй цикл, окружающий S_1, помечается как DOSEQ, а третий — как DOALL. Сильно связная компонента, включающая S_2, содержит одну дугу на уровне 2 и не содержит дуг на уровне 3. Следовательно, второй цикл, окружающий S_2, помечается как DOALL, а третий — как DOSEQ. В итоге мы получаем:

```
DOSEQ i = 1, n
DOSEQ j = 1, n
      DOALL k = 1, n
      S2: b(i, j, k) = b(i, j-1, k+j) + a(i-1, j, k)
  ENDDO
ENDDO
DOALL j = 1, n
  DOSEQ k = 1, n
      S1: a(i, j, k) = a(i-1, j+i, k) + a(i, j, k-1) + b(i, j-1, k)
  ENDDO
ENDDO
ENDDO
```

4.2. Сильные и слабые стороны алгоритма

В работе [32] было показано, что для каждого оператора исходного кода алгоритм Аллена—Кеннеди обнаруживает столько объемлющих параллельных циклов, сколько вообще возможно. Более точно, рассмотрим оператор S исходного кода, и пусть L_i — один из объемлющих циклов. Тогда L_i будет помечен как параллельный только в том случае, когда не существует зависимости на уровне i между двумя экземплярами S. Однако этот результат показывает только то, что алгоритм Аллена—Кеннеди является оптимальным среди тех распараллеливающих алгоритмов, которые рассматривают в преобразованном коде экземпляры S в точно тех же циклах, что и в исходном коде. В действительности можно доказать намного более сильное следующее утверждение [42].

Теорема 1. *Алгоритм Аллена—Кеннеди является оптимальным среди всех алгоритмов нахождения параллелизма, которые принимают СУГЗ в качестве входа.*

В работе [43] доказано, что для любого гнезда циклов N_1 существует гнездо циклов N_2, имеющее тот же СУГЗ, такое, что для любого оператора S из N_1, окруженного после распараллеливания d_S последовательными циклами, в точном графе зависимостей N_2 существует путь зависимости, который включает $\Omega(N^{d_S})$ экземпляров оператора *S*. Другими словами, алгоритм Аллена—Кеннеди не различает гнезд N_1 и N_2, которые имеют один и тот же СУГЗ, и распараллеливающий алгоритм является оптимальным в сильном смысле на гнезде N_2, так как на каждом операторе он достигает верхней

границы параллелизма, определенного самыми длинными путями зависимостей в расширенном графе зависимостей.

Таким образом, доказано, что в случаях, когда вся доступная информация — это та, которая содержится в СУГЗ, не представляется возможным обнаружить больше параллелизма, чем находится алгоритмом Аллена—Кеннеди. Другими словами, алгоритм Аллена—Кеннеди хорошо адаптирован к представлению зависимостей в виде уровней. Следовательно, чтобы обнаружить большее количество параллелизма, чем этот алгоритм, требуется иметь больше информации о зависимостях. Классическими примерами того, как можно превзойти алгоритм Аллена—Кеннеди, являются гнездо циклов примера 4, в котором параллелизм выявляется простой перестановкой циклов (рис. 4), и гнездо циклов примера 5, в котором для выявления параллелизма достаточно осуществить преобразования перекоса и перестановки (рис. 5).

Пример 4.

```
DO i=1,n
    DO j=1,n
        a(i,j)=a(i-1,j-1)+a(i,j-1)
    ENDDO
ENDDO
```

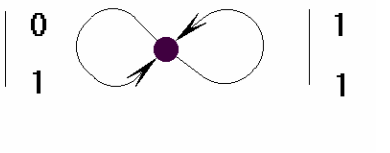


Рис. 4. СГЗ программы примера 4

Пример 5.

```
DO i=1,n
    DO j=1,n
        a(i,j)=a(i-1,j)+a(i,j-1)
    ENDDO
ENDDO
```

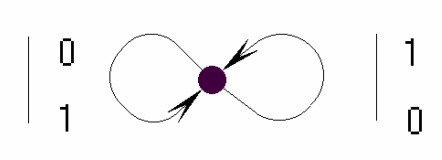


Рис. 5. СГЗ программа примера 5

5. АЛГОРИТМ ВОЛЬФА—ЛЭМА

Фрагменты программ примеров 4 и 5 содержат параллелизм, который не может быть обнаружен с помощью алгоритма Аллена—Кеннеди. Следовательно, как показывает теорема 1, этот параллелизм не может быть извлечен, если зависимости представлены в виде уровней. Чтобы обойти это ограничение, Вольф и Лэм [60] предложили алгоритм, использующий в качестве входа векторы направления. Их работа объединяет все предыдущие алгоритмы, основанные на операциях с элементарными матрицами, такие как перекос цикла, перестановка циклов, обращение цикла, в единую структуру: структуру допустимых *унимодулярных преобразований.*

5.1. Цель

Цель, которую преследовали Вольф и Лэм, состояла в построении множеств полностью переставляемых гнезд циклов. Полностью переставляемые гнезда являются основой для всех техник мозаичного размещения [51, 59, 60]. Мозаичное размещение используется для выявления среднезернистого и крупнозернистого параллелизма. Более того, множество d полностью переставляемых циклов может быть переписано в виде одного последовательного цикла и d-1 параллельных циклов. Следовательно, этот метод также может быть использован для выявления мелкозернистого параллелизма.

Алгоритм Вольфа—Лэма строит максимальное множество самых внешних полностью переставляемых[5] циклов. Затем он рекурсивно просматривает оставшиеся размерности и зависимости, не задействованные этими цик-

[5] i-й и i+1-й циклы являются переставляемыми в том и только том случае, когда i-я и i+1-я компоненты любого вектора расстояния глубины ≥ i неотрицательны.

лами. Версия, представленная в [60], строит множество циклов с помощью анализа более простых случаев, полагаясь на эвристики для гнезд циклов с глубиной 6 и более. В оставшейся части данного раздела мы рассмотрим этот алгоритм с теоретической точки зрения и дадим его наиболее общий вариант.

5.2. Теоретическое обоснование

Унимодулярные преобразования обладают двумя важными свойствами: линейностью и обратимостью. Для данного унимодулярного преобразования T свойство линейности позволяет с легкостью определить, является ли T допустимым. Действительно, T является допустимым в том и только том случае, когда $Td >_l 0$ для всех ненулевых векторов расстояния d. Обратимость позволяет легко переписывать код, так как преобразование представляет собой простое изменение базиса в Z^n.

В общем случае справедливость выражения $Td >_l 0$ не может быть проверена для всех векторов *расстояния*, число которых может быть слишком большим. Следовательно, нужно попытаться гарантировать, что $Td >_l 0$ для всех ненулевых векторов *направления*, с обычными математическими соглашениями, в $Z \cup \{*\} \cup (Z \times \{+, -\})$. В дальнейшем мы ограничимся рассмотрением только ненулевых векторов направления, известных как *лексикографически положительные* векторы направлений.

Обозначим через t(1), ..., t(n) строки T. Пусть Γ — замыкание конуса, порождаемого всеми векторами направления. Для вектора направления d:

$$Td >_l 0 \Leftrightarrow \exists k_d,\ 1 \le k_d \le n \mid\ \forall i,\ 1 \le i \le k_d,\ t(i).d = 0 \text{ и } t(k_d).d > 0.$$

Это означает, что зависимости, представленные вектором d, порождаются в цикле уровня k_d. Если $k_d = 1$ для всех векторов расстояния d, тогда все зависимости порождаются в первом цикле, а все внутренние циклы являются циклами DOALL. Тогда t(1) называется *вектором синхронизации* или *разделяющей гиперплоскостью*. Такой вектор синхронизации существует только в том случае, когда Γ является направленным, т. е. тогда и только тогда, когда Γ не содержит линейного пространства. Это также эквивалентно тому факту, что конус

$$\Gamma^+ = \{y \mid \forall x \in \Gamma,\ y.x \le 0\}$$

является полномерным. Построение T по n линейно независимым векторам из Γ^+ позволяет преобразовывать циклы в n полностью переставляемых циклов.

Понятие вектора синхронизации лежит в основе метода гиперплоскостей и его вариаций (см., например, [37, 55]), особенно полезных при выявлении мелкозернистого параллелизма, тогда как понятие полностью переставляемых циклов является базисом всех техник мозаичного размещения. Как было сказано ранее, обе формулировки эквивалентны для случая Γ^+.

Когда конус Γ не является направленным, Γ^+ имеет размерность r, $1 \le r \le n$, $r = n - s$, где s — размерность пространства линейности Γ. При наличии r линейно независимых векторов в Γ^+, можно преобразовать гнездо циклов таким образом, что r самых внешних циклов будут полностью переставляемыми. Затем можно рекурсивно применять ту же технику для преобразования n-r внутренних циклов, рассматривая векторы направления, не задействованные еще ни одним из r внешних циклов, т. е. рассматривая векторы направления, входящие в пространство линейности Γ. Это основная идея алгоритма Вольфа—Лэма [60], представленного ниже.

5.3. Обобщенный алгоритм

Алгоритм Вольфа—Лэма воспринимает на входе множество векторов направления D и последовательность линейно независимых векторов E (первоначально пустую), из которых он строит матрицу преобразования с помощью следующей процедуры:

проц WOLF-LAM (D, E : **множество векторов**)=
Определить Γ как замыкание конуса, генерируемого векторами направления из D;
Определить $\Gamma^+ = \{y \mid \forall x \in \Gamma, y.x \le 0\}$, и пусть r — размерность Γ^+;
Дополнить E до множества E' r линейно независимых векторов из Γ^+ (по построению $E \subset \Gamma^+$);
Пусть D' — подмножество D, определяемое по правилу: $d \in D \Leftrightarrow \forall v \in E'$, $v.d = 0$ (то есть $D' = D \cap E'^{\perp} = D \cap$ lin.space(Γ));
Вызвать WOLF-LAM(D', E')
все.

Поскольку процесс, описываемый данной процедурой, может дать в итоге неунимодулярную матрицу, построение желаемой унимодулярной

матрицы T может быть проведено выполнением следующей последовательности шагов.

1. Взяв в качестве D множество векторов направления, а в качестве E пустое множество ∅, вызвать WOLF-LAM(D, E).
2. Построить невырожденную матрицу T_1, первые строки которой являются векторами из построенного множества E (в том же порядке). Пусть $T_2 = pT_1^{-1}$, где p выбирается таким образом, что T_2 будет целочисленной матрицей.
3. Вычислить левую эрмитову форму T2, T2 = QH, где H — неотрицательная нижняя треугольная матрица, а Q — унимодулярная матрица.
4. Q^{-1} есть искомая матрица преобразования (поскольку $pQ^{-1}D = HT_1D$).

Проиллюстрируем работу описанного алгоритма на следующем гнезде циклов.

Пример 6.

```
DO i = 1, n
DO j = 1, n
    DO k = 1, n
        a(i, j, k) = a(i-1, j+i, k) + a(i, j, k-1) + a(i, j-1, k+1)
    ENDDO
ENDDO
ENDDO
```

Множество векторов расстояния для данного гнезда равно D = {(1, –, 0), (0, 0, 1), (0, 1, -1)} (см. рис. 6).

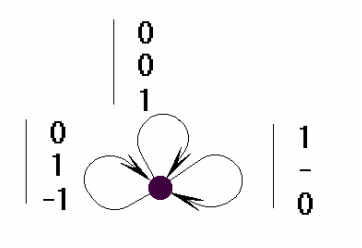


Рис. 6. СГЗ программы примера 6

Нетрудно видеть, что для примера 6 пространство линейности Γ(D) является двумерным (оно порождается векторами (0, 1, 0) и (0, 0, 1)). Следовательно, $\Gamma^+(D)$ является одномерным и порождается $E_1 = \{(1, 0, 0)\}$. Тогда $D' = \{(0, 0, 1), (0, 1, -1)\}$, и Γ(D') является направленным. E_1 пополняется двумя векторами из $\Gamma^+(D')$, например, $E_2 = \{(0, 1, 0\}, (0, 1, 1)\}$. В данном конкретном случае матрица преобразования, имеющая строки E_1 и E_2, уже является унимодулярной и соответствует простому перекосу цикла. Для выявления DOALL-циклов мы выбираем первый вектор E_2 в относительно внутренней части Γ^+, например, $E_2 = \{(0,2,1), (0,1,0)\}$. В терминах преобразований циклов это сводится к перекосу цикла k со множителем 2 и последующей перестановке циклов j и k:

```
DOSEQ i = 1, n
DOSEQ k = 3, 3×n
    DOALL j = max(1,⌈(k-n)/2⌉), min (n,⌊(k-1)/2⌋)
        a(i, j, k-2×j)=a(i-1, j+i, k-2×j)+a(i, j, k-2×j-1)+a(i, j-1, k-2×j+1)
    ENDDO
ENDDO
ENDDO
```

5.4. Сильные и слабые стороны алгоритма

Вольф и Лэм показали оптимальность этой методологии (теорема B.6 из [60]): «алгоритм, который обнаруживает максимальный крупнозернистый параллелизм, а затем рекурсивно вызывает себя на внутренних циклах, производит параллелизм максимально возможной степени». Однако и в этом случае оптимальность следует понимать в контексте используемого анализа: а именно — наличие векторов расстояния при полном отсутствии какой-

либо информации о структуре графа зависимостей. Поэтому корректной является следующая формулировка данного утверждения [36].

Теорема 2. *Алгоритм Вольфа—Лэма является оптимальным среди всех алгоритмов нахождения параллелизма, принимающих на входе множество векторов расстояния (неявно предполагается, что гнездо циклов имеет только один оператор или все операторы образуют атомарный блок).*

Таким образом, как и для алгоритма Аллена—Кеннеди, частичная оптимальность алгоритма Вольфа—Лэма в общем случае имеет причиной не методологию алгоритма, но слабость его входного представления: то, что структура СГЗ не принимается во внимание, может повлечь потерю параллелизма. Например, в противоположность алгоритму Аллена—Кеннеди, алгоритм Вольфа—Лэма не находит параллелизма в примере 3 (СГЗ которого приведен на рис. 7) из-за типичной структуры векторов направления (1, –, 0), (0, 1, –), (0, 0, 1).

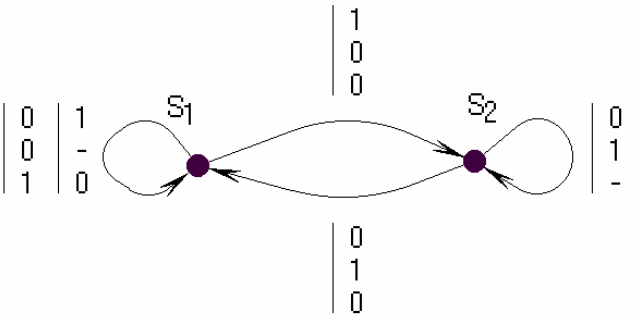


Рис. 7. СГЗ с векторами направлений для программы из примера 3

6. АЛГОРИТМ ДАРТЕ—ВИВЬЕНА

В этом разделе рассматривается третий распараллеливающий алгоритм, который принимает на входе многогранные сокращенные графы зависимостей. Вначале мы изложим некоторую мотивацию необходимости нового алгоритма (разд. 6.1), а затем приведем пошаговое описание алгоритма и обсудим его работу на примерах.

6.1. Потребность в новом алгоритме

Рассмотренные два распараллеливающих алгоритма таковы, что каждый из них может дать на выходе чисто последовательный код для той же самой программы, в которой другой алгоритм обнаружит некоторый параллелизм. Это побуждает к поискам нового алгоритма, объединяющего достоинства алгоритмов Вольфа—Лэма и Аллена—Кеннеди. Чтобы достичь такой цели, можно представить себе комбинацию алгоритмов, которая одновременно использует структуру СГЗ и структуру векторов направления, осуществляя следующую последовательность шагов.

1. Вначале он строит конус, генерируемый векторами направления, и преобразует гнездо циклов таким образом, чтобы выявить максимальное внешнее полностью переставляемое гнездо циклов.
2. Затем рассматривает подграф СГЗ, образованный векторами направления, которые не включены в самые внешние циклы, и вычисляет его сильно связные компоненты.
3. Наконец, применяет распределение цикла для разделения этих компонент и рекурсивно применяет эту же технику к каждой компоненте.

Такая стратегия позволяет выявить большее количество параллелизма, комбинируя унимодулярные преобразования и распределение циклов. Однако она не является оптимальной, что демонстрирует пример 7. Действительно, в программе этого примера вышеописанное сочетание алгоритмов Аллена—Кеннеди и Вольфа—Лэма позволяет определить только одну ступень параллелизма, так как на второй фазе СГЗ остается сильно связным. Результат не лучше, чем у базового алгоритма Аллена—Кеннеди. Однако можно обнаружить две ступени параллелизма, назначив $S_1(i, j, k)$ на момент времени $4i - 2k$ и $S_2(i, j, k)$ на момент времени $4i - 2k + 3$.

Пример 7.

```
DO i = 1, n
DO j = 1, n
   DO k = 1, n
      S1: a(i, j, k) = b(i-1, j+i, k) + b(i, j, k+2)
      S2: b(i, j, k) = a(i, j-i, k+j) + a(i, j, k-1)
   ENDDO
ENDDO
ENDDO
```

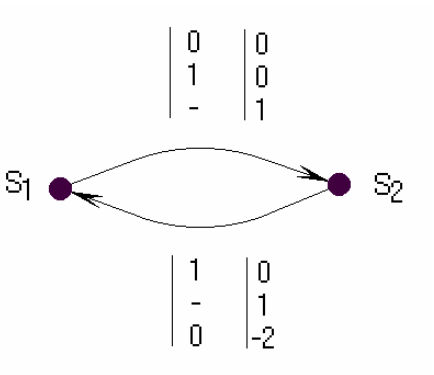


Рис. 8. СГЗ программы примера 7

Нам хотелось иметь простой распараллеливающий алгоритм, который бы находил некоторый параллелизм хотя бы во всех тех случаях, в которых это может сделать алгоритм Аллена—Кеннеди или алгоритм Вольфа—Лэма. Естественным решением было бы применить сначала алгоритм Аллена—Кеннеди, а затем алгоритм Вольфа—Лэма (или комбинацию обоих алгоритмов) и выдать наилучший результат. Но такой наивный подход не будет достаточно мощным, поскольку он использует либо структуру графа зависимостей, либо векторы направления, но не в состоянии воспользоваться знанием обоих структур одновременно. К примеру, предложенная комбинация двух алгоритмов могла бы использовать структуру графа зависимостей до или после вычисления максимального множества полностью переставляемых циклов, но никогда — во время этого вычисления. Утверждается [36], что и графовая структура, и векторы направления *должны* быть использованы одновременно — по той причине, что ключевым понятием при планировании СГЗ является не конус, генерируемый векторами направления (т. е. весами дуг СГЗ), а конус, генерируемый *весами циклов* СГЗ.

Все вышеописанное является мотивацией создания многомерного планирующего алгоритма, представленного ниже. Его можно рассматривать как комбинацию унимодулярных преобразований, распределения цикла и метода сдвига индексов. Это алгоритм, объединяющий достоинства алгоритмов Аллена—Кеннеди и Вольфа—Лэма, был предложен Дарте и Вивьеном [42]. Прежде чем переходить к его описанию, поясним выбор представления зависимостей, используемый алгоритмом.

6.2. Многогранные зависимости: пример

В данном разделе будет приведен пример гнезда циклов, содержащего параллелизм, который не может быть обнаружен, если зависимости представлены в виде уровней или векторов направления. Вместе с тем, пример таков, что для нахождения параллелизма в этом гнезде циклов нет необходимости использовать точное представление зависимостей, а достаточно иметь представление зависимостей в виде многогранника.

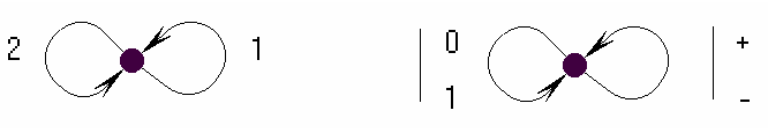
Рассмотрим пример 8. На рис. 9 представлены точные зависимости для этого фрагмента кода, а на рис. 10 изображены соответствующие (сокращенные) графы, дуги зависимостей которых помечены уровнями зависимостей и векторами направления, соответственно. Рассмотрим, что получается в результате применения к данному гнезду циклов рассмотренных ранее алгоритмов.

Пример 8.

```
DO i = 1, n
DO j = 1, n
    S : a(i, j) = a(j, i) + a(i, j-1)
ENDDO
ENDDO
```

$$S(i, j) \xrightarrow{flow} S(i, j+1) \text{ для } n \geq i \geq 1,\ n \geq j \geq 1,$$
$$S(i, j) \xrightarrow{flow} S(j, i) \text{ для } n \geq j > i \geq 1,$$
$$S(j, i) \xrightarrow{anti} S(i, j) \text{ для } n \geq i > j \geq 1,$$

Рис. 9. Точные отношения зависимостей для гнезда циклов из примера 8



а *б*

Рис. 10. СГЗ для фрагмента программы примера 8, в котором дуги помечены уровнями зависимостей (а) и векторами направления (б)

Алгоритм Аллена—Кеннеди базируется на рассмотрении уровней зависимостей. В программе примера 8 уровни трех зависимостей равны соответственно 2, 1 и 1. Имеется цикл зависимостей на уровне 1 и на уровне 2. Таким образом, параллелизм не обнаруживается.

Алгоритм Вольфа—Лэма рассматривает векторы зависимостей. В программе примера 8 векторы зависимостей выглядят как (0, 1), (+, –) и (+, –). Во втором измерении «1» и «–« не позволяют обнаружить два полностью переставляемых цикла. Следовательно, код остается без изменения и параллелизм не обнаруживается.

Для сравнения рассмотрим также результат применения алгоритма Фотрие, который будет описан в разд. 7. Алгоритм воспринимает в качестве входа точные зависимости и на выходе дает допустимый план $T(i, j) = 2i + j - 3$. Таким образом, алгоритм обнаруживает один уровень параллелизма.

Для данного конкретного гнезда циклов представление зависимостей в виде уровней или векторов направления не является достаточно точным для обнаружения параллелизма. В этом и состоит причина неудачи первых двух алгоритмов. Точный анализ зависимостей, связанный с методами линейного программирования, требующими решения больших[6] параметрических линейных программ (как в алгоритме Фотрие), позволяет обнаружить одну степень параллелизма. Соответствующий распараллеленный код будет выглядеть следующим образом:

```
DO j = 3, 3n
  DOPAR i = max (1, ⌈(j-n)/2⌉), min (n, ⌊(j-1)/2⌋)
        a(i, j-2i) = a(j-2i, i) + a(i, j-2i-1)
  ENDDO
ENDDO
```

Однако для гнезда циклов примера 8 точное представление зависимостей вовсе не является необходимым условием для обнаружения параллелизма. Действительно, можно заметить, что имеется униформная зависимость $u = (0, 1)$ и множество векторов расстояний $\{(j - i, i - j) = (j - i)(1, -1) \mid 1 \le j - i \le n - 1\}$, которое может быть приближено (сверху) множеством $P = \{(1, -1) + \lambda(1, -1) \mid \lambda \ge 0\}$. P является многогранником с одной вершиной $v = (1, -1)$ и одним лучом $r = (1, -1)$. Предположим теперь, что мы ищем линей-

[6] Количество неравенств и переменных связано с количеством граничных условий, которые определяют область допустимых значений каждого отношения зависимости.

ный план $T(i, j) = x_1 i + x_2 j$. Пусть $X = (x_1, x_2)$. Чтобы план T был допустимым, нужен такой X, чтобы выполнялось $Xd \geq 1$ для любого вектора зависимости d. Следовательно, $X(0, 1) \geq 1$ и $Xp \geq 1$ для всех $p \in P$. Последнее неравенство эквивалентно следующему: $X(1, -1) + \lambda X(1, -1) \geq 1$ при $\lambda \geq 0$, что эквивалентно $X(1, -1) \geq 1$ и $X(1, -1) \geq 0$, т. е. $Xv \geq 1$ и $Xr \geq 0$. Следовательно, достаточно решить следующие три неравенства:

$Xu \geq 1$ $\quad$ $Xv \geq 1$ $\quad$ $Xr \geq 0$, т.е.

$$X\begin{pmatrix}0\\1\end{pmatrix} \geq 1 \qquad X\begin{pmatrix}0\\-1\end{pmatrix} \geq 1 \qquad X\begin{pmatrix}-0\\1\end{pmatrix} \geq 0$$

что ведет, как у Фотрие, к $X = (2, 1)$. Таким образом, в данном примере приближение зависимостей с помощью уровней или даже векторов направления не является достаточным для определения параллелизма. Однако с помощью приближенного представления зависимостей в виде многогранника можно обнаружить тот же параллелизм, что и с помощью точного анализа зависимостей, но при этом решая более простое множество уравнений.

Особенно важна здесь «униформизация», которая позволяет нам перейти от неравенства на многограннике P к униформным неравенствам на v и r. Благодаря ей исчезают аффинные ограничения, и нам нет необходимости использовать аффинную форму леммы Фаркаша, как в алгоритме Фотрие (см. разд. 7). Чтобы лучше понять принцип «униформизации», будем рассуждать в терминах путей зависимостей. Рассмотрим дугу e, ведущую из оператора S к оператору T и помеченную вектором расстояния $p = v + \lambda r$, как путь φ, который использует один раз «униформный» вектор зависимости v и λ раз «униформный» вектор зависимости r. Такой способ рассмотрения представлен на рис. 11, где введена новая вершина S', которая позволяет моделировать φ и дугу нулевого веса, ведущую из S' обратно в начальную вершину T. Принцип «униформизации» является основной идеей алгоритма распараллеливания циклов, описанного в данном разделе.

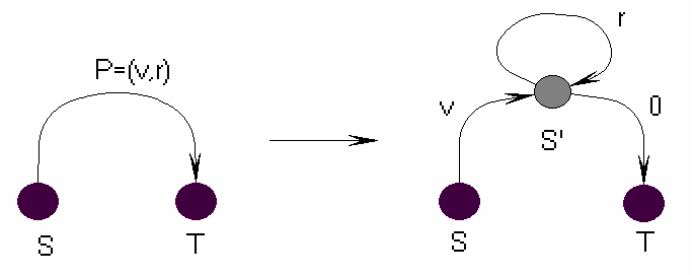


Рис. 11. Моделирование дуги, помеченной многогранником, с одной вершиной и одним лучом

«Униформизуя» зависимости, мы фактически «униформизовали» граничные условия и свели исходную проблему планирования с аффинными граничными условиями к простой проблеме планирования, где все зависимости являются униформными (u, v и r). Однако есть два фундаментальных различия между этой структурой и классической структурой униформного гнезда циклов.

- Униформные векторы зависимостей не обязательно являются лексически положительными (например, луч может быть равен (0, -1)). Поэтому задача планирования будет более сложной. Однако она может быть решена с помощью техник, подобных используемым при решении задачи нахождения решений систем униформных рекуррентных уравнений [52].
- Ограничения, налагаемые на луч r, являются более слабыми, чем в классическом случае, поскольку вместо $Xr \geq 1$ требуется, чтобы $Xr \geq 0$. Это ослабление должно приниматься во внимание распараллеливающим алгоритмом.

6.3. Работа алгоритма: пример

Рассмотрим пример, иллюстрирующий работу алгоритма. Будем считать, что в сокращенном графе зависимостей дуги помечены векторами направления. Граф зависимостей, изображенный на рис. 12, был построен с помощью анализатора зависимостей Tiny [62].

Пример 9.

```
DO i = 1, n
DO j = 1, n
    DO k = 1, n
        a(i, j, k) = c(i, j, k-1) + 1
        b(i, j, k) = a(i-1, j+i, k) + b(i, j-1, k)
        c(i, j, k+1) = c(i, j, k) + b(i, j-1, k+i) + a(i, j-k, k+1)
    ENDDO
ENDDO
ENDDO
```

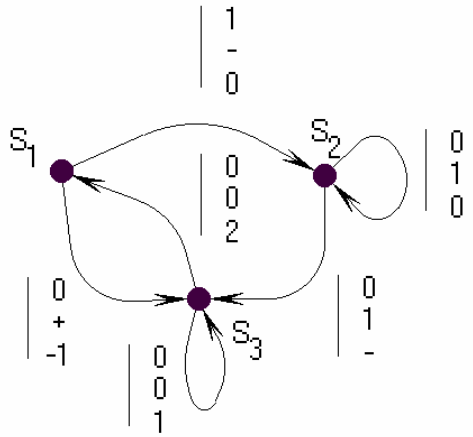


Рис. 12. СГЗ программы примера 9

Нетрудно убедиться, что алгоритмы Аллена—Кеннеди и Вольфа—Лэма не способны найти весь содержащийся в этой программе параллелизм: третий оператор кажется полностью последовательным. Однако алгоритм обнаружения параллелизма, который будет представлен в следующем разделе, способен построить следующий многомерный план: (2i + 1, 2k) для первого оператора, (2i, j) для второго и (2i + 1, 2k + 3) для третьего. Этот план соответствует коду с явным параллелизмом, приведенному ниже (в котором,

однако, не было проведено никаких модификаций наподобие отшелушивания цикла с целью устранения операторов IF). Таким образом, для каждого оператора может быть обнаружен один уровень параллелизма.

```
DOSEQ i = 1, n
DOSEQ j = 1, n
  DOPAR k = 1, j
       b(i, j, k) = a(i-1, j+i, k) + b(i, j-1, k)
  ENDDO
ENDDO
DOSEQ k = 1, n+1
  IF (k ≤ n) THEN
       DOPAR j = k, n
        a(i, j, k) = c(i, j, k-1) + 1
       ENDDO
  ENDIF
  IF (k ≤ 2) THEN
       DOPAR j = k-1, n
        c(i, j, k) = c(i, j, k-1) + b(i, j-1, k+i+1) + a(i, j-k+1, k)
       ENDDO
  ENDIF
ENDDO
ENDDO
```

Этот код был сгенерирован из вышеприведенного плана с помощью процедуры “codegen” программы Omega Calculator[7], поставляемого в комплекте Petit [53]. Следует напомнить, что вышеприведенный код является «виртуальным» в том смысле, что алгоритм только выявляет скрытый параллелизм. Реальный код не обязательно должен быть именно таким.

6.4. Шаг униформизации

Вначале мы покажем, как МСГЗ (многогранные сокращенные графы зависимостей) могут быть изображены с помощью эквивалентной, но более простой в обращении структуры — униформных графов зависимостей, то есть графов, дуги которых помечены константными векторами зависимо-

[7] Omega Calculator служит для вычисления зависимостей, проверки допустимости программных преобразований и преобразования программ заданным образом.

стей. Такая униформизация достигается с помощью приведенного ниже *алгоритма трансляции*.

Чтобы избежать путаницы между вершинами графа зависимостей и вершинами многогранника зависимостей, мы будем называть первые *узлами*, а вторые — *вершинами*. Будут использоваться также следующие соглашения. Исходный МСГЗ, описывающий зависимости в распараллеливаемом коде, называется *исходным графом* и обозначается как $G_0 = (V, E)$. Униформный СГЗ, эквивалентный G_0 и построенный с помощью алгоритма трансляции, называется *униформным графом* или *трансляцией* G_0 и обозначается как $G_u = (W, F)$.

Алгоритм трансляции строит G_u путем сканирования всех дуг G_0. Он начинает с $G_u = (W, F) = (V, \varnothing)$, и для каждой дуги e из E добавляет в G_u новые узлы и новые дуги в зависимости от вида многогранника P(e). Будем называть вновь созданные узлы *виртуальными узлами* в отличие от *реальных узлов*, соответствующих узлам G_0.

Пусть e — дуга из E. Обозначим через x_e и y_e соответственно начало и конец дуги e, т. е. узел, из которого e исходит, и узел, в который она заходит. Это определение обобщается до путей: начало (соответственно конец) пути — это начало (соответственно конец) его первой (соответственно последней) дуги.

Будем следовать нотациям, введенным в разделе 3.2: ω, ρ и λ обозначают количество вершин v_i, лучей r_i и линий l_i многогранника P(e).

Трансляцию осуществляет следующий алгоритм:

начало
Пусть W = V и F = $\varnothing$;
для всех e: $x_e \rightarrow y_e \in E$ **цикл**
Добавить к W новый виртуальный узел n_e;
Добавить к F ω дуг с весами $v_1, v_2, ..., v_\omega$, соединяющих x_e с n_e;
Добавить к F ρ петель, соединяющих n_e, с весами $r_1, r_2, ..., r_\rho$;
Добавить к F λ петель, соединяющих n_e, с весами $l_1, l_2, ..., l_\lambda$;
Добавить к F λ петель, соединяющих n_e, с весами $-l_1, -l_2, ..., -l_\lambda$;
Добавить к F дугу с нулевым весом, ведущую из n_e в y_e.
все
конец.

Вернемся к программе примера 9. Граф МСГЗ данной программы изображен на рис. 12. Рис. 13 представляет связанный с ней униформный граф зависимостей. Граф имеет три виртуальных узла, соответствующих символу

«+» и двум символам «–» в исходных векторах направления; это узлы, помеченные символами S'_1, S'_2 и S'_3.

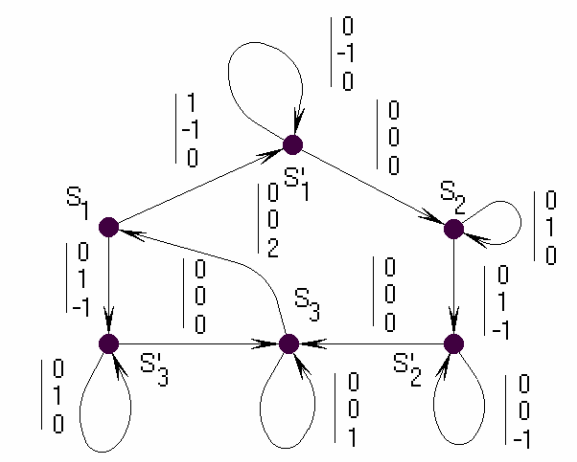


Рис. 13. Транслированный униформный сокращенный граф зависимостей

6.5. Шаг планирования

Шаг планирования принимает на входе транслированный граф зависимостей G_u и строит многомерный план для каждого реального узла, т. е. для каждого узла G_u, который соответствует узлу G_0. G_u предполагается сильно связным (в противном случае нужно вызывать алгоритм для каждой сильно связной компоненты G_u).

Алгоритм является рекурсивным. Каждый шаг рекурсии строит определенный подграф G' текущего обрабатываемого графа G. Как только G' построен, выводится множество линейных ограничений и может быть вычислен допустимый план, охватывающий все дуги зависимостей, не входящие в G'. Затем алгоритм продолжает работу с оставшимися дугами, то есть дугами из G' (более точно, с дугами из G' и некоторыми добавочными дугами — см. ниже).

G' определяется как подграф G, генерируемый всеми дугами G, принадлежащими по меньшей мере к одному мультициклу нулевого веса. Мультицикл есть объединение циклов, не обязательно связанных, а его вес есть

сумма весов составляющих его циклов. G' строится на основе решения линейной программы (см. разд. 6.6).

Шаг планирования описывается нижеприведенным рекурсивным алгоритмом. Первым вызовом является DARTE-VIVIEN(G_u, 1). Для каждого реального узла S из G_u алгоритм строит последовательность векторов X^1_S, ..., X^{dS}_S и последовательность констант ρ^1_S, ..., ρ^{dS}_S, которые определяют допустимый многомерный план.

проц DARTE-VIVIEN(G: **граф**, k: **целое**)=

1. Построить G' — такой подграф G, который порожден всеми дугами, принадлежащими по меньшей мере одному мультициклу нулевого веса в G;
2. Добавить к G' все дуги, ведущие из реального узла x_e к виртуальному узлу y_e, и все петли, соединяющие y_e, если дуга $e = (x_e, y_e)$ уже входит в G';
3. Выбрать вектор X и константу ρ_S для каждого узла S из G таким образом, чтобы выполнялось два условия:

$$e = (x_e, y_e) \in G' \text{ или } x_e \text{ — виртуальный узел} \Rightarrow$$
$$X\omega(e) + \rho_{y_e} - \rho_{x_e} \geq 0$$

$$e = (x_e, y_e) \notin G' \text{ или } x_e \text{ — реальный узел} \Rightarrow$$
$$X\omega(e) + \rho_{y_e} - \rho_{x_e} \geq 1,$$

   и для всех реальных узлов S из G положить $\rho^k_S = S$ и $X^k_S = X$.
4. **если** G' — пустой граф или содержит только виртуальные узлы
   то возврат все;
5. **если** G' — сильно связный граф и содержит реальные узлы
   то G является невычислимым (а исходный МСГЗ
   G_0 — несогласованным); **возврат**
   все;.
   Провести декомпозицию G' на сильно связные компоненты G_i;
   для всех G_i, имеющих реальные узлы, **цикл**
   DARTE-VIVIEN(G_i, k+1)
   все.

все.

К данному описанию алгоритма можно сделать следующие два замечания.

1. Шаг (2) является необходимым только для МСГЗ общего вида: его можно опустить, к примеру, для СГЗ, помеченных векторами направления (см. [44] для более подробного изложения). В этом случае решение простой линейной программы может заменить одновременно шаг (1) и шаг (3).
2. На шаге (3) мы сознательно не указываем, каким образом выбираются вектор X и константы ρ, что позволяет применять различные критерии выбора. Например, можно выбрать максимальное множество линейно независимых векторов X, если целью является построение полностью переставляемых циклов (см. [46]).

Вернемся к примеру 9. Рассмотрим униформный граф зависимостей на рис. 13. Имеются два элементарных цикла с весами (1, 0, 1) и (0, 1, 1) и пять петель с весами (0, 0, 1), (0, 0, –1), (0, 1, 0) (дважды) и (0, –1, 0). Следовательно, все дуги (кроме дуг, принадлежащих только к циклу с весом (1, 0, 1)) принадлежат к мультициклу нулевого веса. Подграф G' показан на рис. 14.

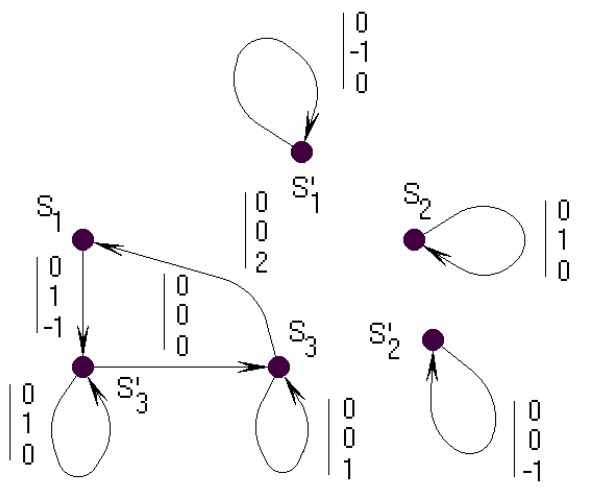


Рис. 14. Подграф мультициклов нулевого веса для примера 9

Ограничения, следующие из дуг подграфа G', устанавливают, что X = (x, y, z) должен быть ортогонален к весам всех циклов G'. Следовательно, y = z = 0. Наконец, рассматривая остальные ограничения, мы находим решение: X = (2, 0, 0), $\rho_{S1} = \rho_{S3} = 1$ и $\rho_{S2} = 0$. В G' остаются четыре сильно связных компоненты, две из которых не рассматриваются, так как имеют только виртуальные узлы. Две оставшиеся компоненты не содержат мультициклов нулевого веса. Сильно связная компонента с единственным узлом S_2 может быть спланирована с вектором X = (0, 1, 0), тогда как исследование второй сильно связной компоненты приводит к одному из возможных решений: X = (0, 0, 2), $\rho_{S1} = 0$ и $\rho_{S3} = 3$.

Суммируя все результаты, мы находим уже рассмотренные в разд. 6.3. двумерные планы: (2i, j) для S_2, (2i + 1, 2k) для S_1 и (2i + 1, 2k + 3) для S_3.

6.6. Схематические пояснения

Граф G_u не всегда соответствует СГЗ гнезда циклов, так как его векторы зависимостей не обязательно лексически неотрицательны. Фактически, если забыть о том, что некоторые узлы виртуальны, то G_u — это сокращенный граф зависимостей *Системы Униформных Рекуррентных Уравнений*, введенной Карпом, Миллером и Виноградом в [52]. Карп, Миллер и Виноград исследовали проблему вычислимости таких систем. Они показали, что ее вычислимость связана с проблемой нахождения циклов нулевого веса в ее СГЗ G, что может быть проделано посредством рекурсивной декомпозиции графа, основанной на нахождении мультициклов нулевого веса. Ключевой структурой их алгоритма является G', подграф G, генерируемый дугами, принадлежащими к мультициклу нулевого веса.

G' может быть успешно построен с помощью решения простой линейной программы (программа примера 7 или двойственная ей программа примера 8). Это решение позволяет построить алгоритм распараллеливания, принцип которого двойственен алгоритму Карпа, Миллера и Винограда:

$$\min\left\{\sum_e v_e \quad : \quad q \ge 0,\ v \ge 0,\ w \ge 0,\ q + v = 1 + w,\ Bq = 0\right\}, \tag{4}$$

$$\max\left\{\sum_e z_e \quad : \quad z \ge 0,\ 0 \le z_e \le 1,\ Xw(e) + \rho_{y_e} - \rho_{x_e} \ge z_e\right\}, \tag{5}$$

где w(e) — вектор зависимостей, связанный с дугой e, B = $[CW]^t$, C — матрица связей, а W — матрица векторов зависимостей.

Если не вдаваться в детали, X — это n-мерный вектор, имеется одна переменная ρ для каждой вершины СГЗ и одна переменная z для каждой дуги СГЗ. Дугами G' (или G\G') являются дуги $e = (x_e, y_e)$, для которых $z_e = 0$ (соответственно $z_e = 1$) в оптимальном решении двойственной программы 8, и, эквивалентно этому, для которых $v_e = 0$ (соответственно $v_e = 1$) в изначальной программе 7. Суммируя неравенства $Xw(e) + \rho_{ye} - \rho_{xe} \geq z_e$ в цикле C в G, мы находим, что Xw(C) = 0, если C — цикл из G', и $Xw(C) \geq l(C) > 0$ в противном случае (l(C) — количество дуг C, не принадлежащих G').

Чтобы увидеть связь с алгоритмом Вольфа—Лэма, можно рассмотреть конус Γ, генерируемый *весами циклов* (а не весами дуг); тогда G' — подграф, веса циклов которого генерируют пространство линейности Γ, а X — вектор в относительно внутренней части Γ^+. Однако нет необходимости для построения G' реально строить Γ. Это просто одно из любопытных свойств программ примеров 7 и 8.

Мы обрисовали основные идеи алгоритма Дарте—Вивьена [44]. Требуются некоторые технические модификации для распознавания виртуальных и реальных узлов и для учета природы дуг (помеченных вершинами, лучами или линиями многогранника зависимостей). Отсылаем читателя к работе [42] за полным изложением алгоритма.

6.7. Сильные и слабые стороны алгоритма

Теперь, когда у нас есть многомерный план T, мы можем доказать его оптимальность в терминах степени параллелизма. Можно показать [41, 42], что для каждого оператора S (то есть для каждого узла G_0) количество экземпляров S, исполняемых последовательно в результате применения T, имеет тот же порядок, что и количество экземпляров S, являющихся врожденно последовательными в силу действия зависимостей.

Теорема 3. *Планирующий алгоритм является почти оптимальным, если область итерации содержит полномерный куб размера $\Omega(N)$ (соответственно содержится в кубе размерности $O(N)$) и если d — глубина (количество вложенных рекурсивных вызовов) алгоритма, то время ожидания плана равно $O(N^d)$ и длина самого длинного пути зависимостей равна $\Omega(N^d)$. Более точно, после генерации кода каждый оператор S окружен в точности d_S последовательными циклами, и эти циклы считаются врожденно последовательными также после анализа зависимостей.*

Этот алгоритм также является оптимальным относительно анализа зависимостей. Рассмотрим следующий пример.

Пример 10.

```
DO i = 1, n
DO j = 1, n
      S1: a(i, j) = b(i-1, j+i) + a(i, j-1)
      S2: b(i, j) = a(i-1, j-i) + b(i, j-1)
ENDDO
ENDDO
```

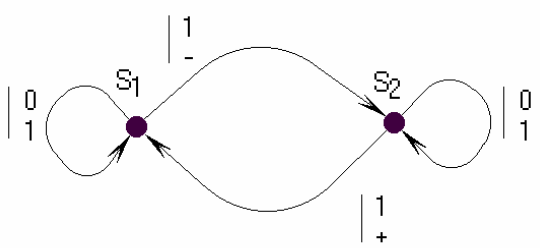


Рис. 15. СГЗ программы примера 10

Если зависимости описываются с помощью векторов расстояния, СГЗ имеет две автозависимости (0, 1) и две дуги, помеченные многогранниками, обе имеющие по одной вершине и одному лучу — (0, 1) и (0, –1), соответственно (см. рис. 15). Следовательно, существует мультицикл нулевого веса. Далее, две реальных вершины принадлежат к G'. Таким образом, глубина алгоритма Дарте—Вивьена равна 2, и никакого параллелизма не обнаруживается.

Однако вычисление итерации (i, j) первым (вторым) оператором на шагу 2i + j (соответственно i + j) приводит к допустимому плану, который обнаруживает одну степень параллелизма[8]. Алгоритм Дарте—Вивьена не способен найти параллелизм в этом примере, так как аппроксимация зависимостей уже потеряла весь параллелизм.

Используемая здесь нами техника нахождения параллельных циклов заключается в поиске многомерных планов, линейные части которых (векто-

[8] Планы $\lfloor(3/2)i + j + 1/2\rfloor$ и $\lfloor(1/2)i + j\rfloor$ минимизируют время ожидания, но написание кода становится более сложным.

ры X) могут быть разными для разных операторов, даже если они принадлежат к одной и той же сильно связной компоненте. Это является основой алгоритма Фотрие [47], математической базой которого является аффинная форма леммы Фаркаша. Однако теорема 3 показывает, что нет необходимости искать различные линейные части (построение которых будет более дорогим и приведет к усложнению процесса переписывания) в заданной сильно связной компоненте текущего подграфа G', если зависимости заданы с помощью векторов расстояния. С другой стороны, пример 10 показывает, что такое усовершенствование может быть полезным только тогда, когда доступен более точный анализ зависимостей.

7. АЛГОРИТМ ФОТРИЕ

В работе [47] Пол Фотрие предложил алгоритм планирования статических управляющих программ с аффинными зависимостями. Этот алгоритм использует точный анализ зависимостей, который всегда является осуществимым для таких типов программ [45]. В этом его отличие от трех ранее рассмотренных алгоритмов (Аллена—Кеннеди, Вольфа—Лэма, Дарте—Вивьена), которые работают с теми или другими аппроксимациями зависимостей.

Алгоритм Фотрие принимает на входе сокращенный граф зависимостей G, в котором дуга e: $S_i \to S_j$ помечена множеством пар (I, J) таких, что $S_j(J)$ зависит от $S_i(I)$. Далее он рекурсивно строит многомерный аффинный план для каждого оператора гнезда циклов.

проц FEAUTRIER(G : **граф**) =
    Провести декомпозицию G на сильно связные компоненты G_i и топологически отсортировать их.
    для всех сильно связных компонент G_i **цикл**
        Найти аффинный план с участием оператора, который налагает неотрицательную задержку на все зависимости и задействован в наибольшем возможном количестве зависимостей;
        Построить множество G'_i незадействованных дуг;
            если $G'_i \neq \varnothing$ **то** FEAUTRIER(G'_i) **все**
    все
все

Данный алгоритм похож на алгоритм Дарте—Вивьена по своей структуре и по выдаваемому результату, а также тем, что оба алгоритма используют линейные программы для построения аффинных планов. Приведем основные свойства, связанные со сравнением данных двух алгоритмов (см. [36]).

1. Алгоритм Дарте—Вивьена способен планировать программы, даже если точный анализ зависимостей не является осуществимым, но дан СГЗ с многогранником зависимостей. Алгоритм Фотрие способен обрабатывать только статические управляющие программы с аффинными зависимостями. В этом смысле первый алгоритм является более мощным. Однако укажет на попытки обобщить подход, реализованный в алгоритме Фотрие, путем ослабления ограничений на вход за счет использования анализа потока данных на нечетких массивах (Fuzzy Array Dataflow Analysis) [33].
2. Когда доступен анализ зависимостей, алгоритм Фотрие становится намного более мощным. Этот алгоритм способен обрабатывать любое множество циклов, описывающих многогранник, даже если циклы не образуют совершенное гнездо. Алгоритм Дарте—Вивьена также может обрабатывать не являющиеся совершенными вложенные гнезда циклов, либо рассматривая каждый блок совершенно вложенных циклов по отдельности, либо искусственно сливая несовершенно вложенные гнезда. Однако согласно теории, этот подход будет менее естественным и менее мощным.
3. Алгоритм Дарте—Вивьена основан на решении линейных программ, подобных тем, что решает алгоритм Фотрие. Единственное (но фундаментальное) различие состоит в том, что первый ищет менее общие аффинные преобразования. Следовательно, на статических управляющих программах с аффинными зависимостями, алгоритм Фотрие всегда находит больше параллелизма, чем алгоритм Дарте—Вивьена (см. пример 10). Однако, несмотря на это различие, результат оптимальности для алгоритма Дарте—Вивьена дает некоторые указания на случаи оптимальности для Фотрие, которые были впервые введены как «жадные эвристики».
4. Алгоритм Фотрие должен использовать аффинную форму леммы Фаркаша для получения линейных программ, чего алгоритм Дарте—Вивьена избегает благодаря своей схеме униформизации. Следовательно, линейные программы алгоритма Фотрие более сложны.
5. Оба алгоритма расширяют действие от мелкозернистого параллелизма до среднезернистого путем поиска полностью переставляемых циклов. Дарте и др. [40] предложили расширение алгоритма Дарте—Вивьена,

которое представляет собой простое обобщение Вольфа—Лэма. Лим и Лэм [57] предложили расширение алгоритма Фотрие, которое находит максимальные множества полностью переставляемых циклов, минимизируя при этом количество синхронизаций, необходимых в распараллеленном коде.

6. Алгоритм Дарте—Вивьена производит настолько регулярные планы, насколько это возможно, чтобы получить возможно более простой код. Действительно, этот алгоритм переписывает код с использованием аффинных планов, но, в отличие от алгоритма Фотрие, эти аффинные планы выбираются таким образом, чтобы возможно большее количество операторов имели одну и ту же линейную часть: после этого генерация кода может рассматриваться как последовательность частичных унимодулярных преобразований и распределений циклов. В результате полученные программы выходят гарантированно более простыми, чем программы, получаемые с помощью алгоритма Фотрие.

В работе [36] приводятся результаты небольшого сравнительного исследования алгоритмов, которое проводилось на четырех примерах. Как и ожидалось авторами, в этих исследованиях сложность алгоритма Дарте—Вивьена была намного ниже, чем у алгоритма Фотрие. Более удивительным оказался тот факт, что оба алгоритма дали один и тот же результат на всех четырех рассмотренных примерах. Разумеется, для получения окончательного заключения о сравнительных характеристиках алгоритмов данного исследования явно не достаточно: необходимо сравнительное исследование алгоритмов на существенно большем числе реальных программ.

На рис. 16 приведен пример кода, который с очевидностью содержит некоторый параллелизм, но не поддается распараллеливанию ни одним из четырех рассмотренных в данной статье алгоритмов.

```
                        DOPAR i = 1, ⌊n/2⌋
                        a(i) = 1 + a(n-i)
DO i = 1, n             ENDDO
   a(i) = 1 + a(n-i)    DOPAR i = ⌊n/2⌋ + 1, n
ENDDO                   a(i) = 1 + a(n-i)
                    ENDDO
```

Рис. 16. Пример исходного (слева) и распараллеленного (справа) кода

8. ЗАКЛЮЧЕНИЕ

В статье рассмотрены основные существующие алгоритмы распараллеливания циклов — одного из наиболее важных реструктурирующих преобразований. Суммируя их свойства, следует отметить следующее: алгоритм Аллена—Кеннеди является оптимальным для представления зависимостей в виде уровней, а алгоритм Вольфа—Лэма — для представления зависимостей в виде векторов направления (но для гнезда циклов с единственным оператором). Ни один из двух алгоритмов не охватывает другой, поскольку каждый использует информацию, которая не может быть использована вторым (графовая структура для первого алгоритма, векторы направления — для второго). Однако оба алгоритма охватываются алгоритмом Дарте—Вивьена, который является оптимальным для любого многогранного представления векторов расстояния. Алгоритм Фотрие охватывает алгоритм Дарте—Вивьена в случаях, когда зависимости могут быть представлены как аффинные, но вопрос об его оптимальности остается открытым.

Рассмотренная классификация алгоритмов распараллеливания циклов имеет практический интерес. Она дает возможность распараллеливающему компилятору выбрать наиболее подходящий алгоритм: при условии, что задан определенный анализ зависимостей, должен выбираться самый простой и дешевый распараллеливающий алгоритм, который остается оптимальным. Это будет именно тот алгоритм, который лучше всего подходит к доступному представлению зависимостей.

СПИСОК ЛИТЕРАТУРЫ

1. Булышева Л.А. Методы и средства оптимизации вычислений для процессоров с VLIW-архитектурой. — Новосибирск, 1993. — (Препр. / РАН. Сиб.отд-ние. ВЦ; N 976).
2. Вальковский В. А. Распараллеливание алгоритмов и программ. Структурный подход. — М.: Радио и связь, 1989.
3. Векторизация программ: теория, методы, реализация: Сб. статей / Под ред. Г.Д. Чинина. — М.: Мир, 1991.
4. Воеводин В. В. Математические основы параллельных вычислений. — М.: МГУ, 1991.
5. Воеводин Вл. В. Теория и практика исследования параллелизма последовательных программ // Программирование. — 1992. — N 3. — С. 38–54.
6. Глушков В.М., Капитонова Ю.В., Летичевский А.А. Алгебра алгоритмов и динамическое распараллеливание последовательных программ // Кибернетика. — 1982. — N 5. — С. 4–10.

7. Дорошенко А.Е. Математические модели и методы организации высокопроизводительных параллельных вычислений. — Киев: Наукова думка, 2000.
8. Дорошенко А.Е. О преобразованиях циклических операторов к параллельному виду // Кибернетика. — 1984. — N 4. — С. 22–28.
9. Евстигнеев В.А., Касьянов В.Н. Оптимизирующие преобразования в распараллеливающих компиляторах // Программирование. — 1996. — N 6. — С. 12–26.
10. Евстигнеев В.А., Мирзуитова И.Л. Анализ циклов: выбор кандидатов на распараллеливание. — Новосибирск, 1999. — (Препр. / ИСИ СО РАН; N 58).
11. Евстигнеев В. Анализ зависимостей: состояние проблемы // Системная информатика. — Новосибирск: Наука, 2000. — Вып. 7. — С.112–173.
12. Иванников В.П., Гайсарян С.С. Особенности систем программирования для векторно-конвейерной ЭВМ // Кибернетика и вычислительная техника. — М.: Наука, 1986. — Вып. 2. — С. 3–17.
13. Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение. — СПб.: БХВ-Петербург, 2003.
14. Касьянов В.Н. Оптимизация программ // Прикладная информатика. — М.: Статистика, 1983. — Вып. 2. — С. 38–76.
15. Касьянов В.Н. Оптимизирующие преобразования программ. — М.: Наука , 1988.
16. Касьянов В.Н. Трансформационный подход к конструированию и оптимизации программ // Смешанные вычисления и преобразование программ. — Новосибирск, 1991. — С. 30–43.
17. Касьянов В.Н. Трансформационные методы и средства конструирования эффективных и надежных программ // Кибернетика и системный анализ. — 1993. — N 2. — С. 30–39.
18. Системное и математическое обеспечение многопроцессорного вычислительного комплекса ЕС / Под ред. Ю.В. Капитоновой. — М.: ВВИА им. Н.Е. Жуковского, 1985.
19. Трантенгерц Э. А. Программное обеспечение параллельных процессов. — М.: Наука, 1987.
20. Французов Ю. А. Обзор методов распараллеливания кода и программной конвейеризации // Программирование. — 1992. — N 3. — С. 16–37.
21. Черняев А.П. Системы программирования для высокопроизводительных ЭВМ. — Т.3. Вычислительные науки. — М., 1990. — С. 1–141. — (Итоги науки и техн. ВИНИТИ АН СССР).
22. Черняев А.П. Программные системы векторизации и распараллеливания Фортран-программ для некоторых векторно-конвейерных ЭВМ (обзор) // Программирование. — 1991. — N 2. — С. 53–68.
23. Allan V.H., Jones R.B., Lee R.M., Allan S.J. Software pipelining // ACM Computing Surveys. — 1996. — Vol. 27, N 3. — P.367–432.
24. Allen J.R., Kennedy K. Automatic translation of Fortran programs to vector form // ACM Trans. Program Lang. Syst. — 1987. —Vol. 9, N 4. — P. 491–542.

25. Allen J.R., Kennedy K. PFC: a program to convert programs to parallel form. — Houston, 1982. — (Tech. Rep. / Dept. of Math. Sciences, Rice University; MASC-TR82-6).
26. Bacon D. F., Graham S. L., Sharp O. J. Compiler transformations for high-performance computing // ACM Computing Surveys. — 1994. — Vol. 26, N 4. — P.345–420.
27. Banerjee U. A theory of loop permutations // Languages and Compilers for Parallel Computing. — MIT Press, 1990. — P. 54–74.
28. Banerjee U. Data dependence in ordinary programs. — Urbana, 1976. — (Tech. Rep. / Univ. Ill., 76-837).
29. Banerjee U. Dependence analysis for supercomputing. —Norwell: Kluwer Academic Publishers, 1988.
30. Bernstein A. J. Analysis of programs for parallel processing. // IEEE Trans. on Electronic Computers. — 1966. — Vol. 15, N 5. — P. 757–763.
31. Bockle G. Exploitation of fine-grain parallelism. — Berlin: Springer-Verlag, 1995. — (Lect. Notes Comput. Sci.; Vol. 942).
32. Callahan D. A Global Approach to Detection of Parallelism: PhD thes. — Dept. of Computer Science, Rice University. — Houston, 1987.
33. Collard J.-F., Barthou D., and Feautrier P. Fuzzy Array Dataflow Analysis // Proc. of 5th ACM SIGPLAN Symp. on Principles and practice of Parallel Programming. — Santa Barbara, CA, 1995.
34. Collard J.-F., Feautrier P., Risset T. Construction of DO loops from systems of affine constraints // Parallel Processing Letters. — 1995. — Vol. 5, N 3. — P. 421–436.
35. Collard J-F. Code generation in automatic parallelizers // Proc. Int. Conf. on Application in Parallel and Distributed Computing. — North Holland, 1994. — P. 185–194.
36. Computer optimizations for scalable parallel systems: languages, compilation techniques, and run time systems / Santosh Pande; Dharma P. Agrawal (ed.). — Berlin: Springer-Verlag, 2001. — (Lect. Notes Comput. Sci.; Vol. 1808).
37. Darte A., Khachiyan L., Robert Y. Linear scheduling is nearly optimal // Parallel Processing Letters. — 1991. — Vol. 1, N 2. — P. 73–81.
38. Darte A., Robert Y. Affine-by-statement scheduling of uniform and affine loop nests over parametric domains // J. Parallel and Distributed Computing. — 1995. — Vol. 29. — P. 43–59.
39. Darte A., Robert Y. Mapping uniform loop nests onto distributed memory architectures // Parallel Computing. — 1994. — Vol. 20. — P. 679–710.
40. Darte A., Silber J.-A., Vivien F. Combining retiming and scheduling techniques for loop parallelization and loop tiling. — ENS-Lyon, 1996. — (Tech. Rep. / LIP; 96-34).
41. Darte A., Vivien F. Automatic parallelization based on multi-dimensional scheduling. — Lyon, 1994. — (Tech. Rep. / LIP; 94-24).
42. Darte A., Vivien F. On the optimality of Allen and Kennedy's algorithm for parallelism extraction in nested loops //. J. of Parallel Algorithms and Application. — 1996. — (Special issue on Optimizing Compilers for Parallel Languages).

43. Darte A., Vivien F. Optimal fine and medium grain parallelism detection in polyhedral reduced dependence graphs. — Lyon, 1996. — (Tech. Rep. / LIP; 96-06).
44. Darte A., Vivien F. Optimal fine and medium grain parallelism detection in polyhedral reduced dependence graphs // Proc. of PACT'96. — Boston: IEEE Computer Society Press, 1996.
45. Feautrier P. Dataflow analysis of array and scalar references // Int. J. Parallel Programming. — 1991. — Vol. 20, N 1. — P. 23–51.
46. Feautrier P. Some efficient solutions to the affine scheduling problem, part I, one-dimensional time // Int. J. Parallel Programming. — 1992. — Vol. 21, N 5. — P. 313–348.
47. Feautrier P. Some efficient solutions to the affine scheduling problem, part II, multi-dimensional time // Int. J. Parallel Programming. — 1992. — Vol. 21, N 6. — P. 389–420.
48. Goff G., Kennedy K., Tseng C.-W. Practical dependence testing // Proc. of the SIGPLAN Conf. on Programming Language Design and Implementation. — SIGPLAN Not. — 1991. —Vol. 26, N 6. — P. 15–29.
49. Irigoin F., Jouvelot P., Triolet R. Semantical interprocedural parallelization: an overview of the PIPS project // Proc. of the 1991 ACM Internat. Conf. on Supercomputing. —Cologne, 1991.
50. Irigoin F., Triolet R. Computing dependence direction vectors and dependence cones with linear systems. — Fontainebleau, 1987. — (Tech. Rep. / Ecole des Mines de Paris; ENSMP-CAI-87-E94).
51. Irigoin F., Triolet R. Supernode partitioning // Proc. 15th Annual ACM Symp. Principles of Programming Languages. — San Diego: CA, 1988. — P. 319–329.
52. Karp R.M., Miller R.E., Winograd S. The organization of computations for uniform recurrence equations // J. of the ACM. — 1967. — Vol. 14, N 3. — P. 563–590.
53. Kelly W., Maslov V., Pugh W., Rosser E., Shpeisman T., and Wonnacott D. New user interface for Petit and other interfaces: user guide. — University of Maryland, 1995.
54. Kong X., Klappholz D., Psarris K. The I test: an improved dependence test for automatic parallelization and vectorization // IEEE Trans. Par. Distr. Syst. — 1991. — Vol. 2, N 3. — P. 342–349.
55. Lamport L. The parallel execution of DO loops // Commun. of the ACM. — 1974. — Vol. 17, N 2. — P. 83–93.
56. Li Z. Array privatization for parallel execution of loops // Proc. of the 1992 Intern. Conf. on Supercomputing. — 1992. — P. 313—322.
57. Lim A.W., Lam M.S. Maximizing parallelism and minimizing synchronization with affine partitions // Parallel Computing. — 1998. — Vol. 24, Iss. 3-4. — P. 445–475.
58. Pugh W. A practical algorithm for exact array dependence analysis // Commun. ACM. — 1992. — Vol. 35, N 8. — P. 102–115.
59. Schreiber R., Dongarra J.J. Automatic blocking of nested loops. — Knoxville, 1990. — (Tech. Rep. / The University of Tennessee; 90-38).

60. Wolf M.E., Lam M.S. A loop transformation theory and an algorithm to maximize parallelism // IEEE Trans. Parallel Distributed Systems. — 1991. — Vol. 2, N 4. — P. 452–471.
61. Wolfe M. High Performance Compilers For Parallel Computing. — Addison-Wesley Publishing Company, 1996.
62. Wolfe M. Optimizing Supercompilers for Supercomputers: PhD thes. — Dept. of Computer Science, University of Illinois. — Urbana-Champaign, 1982.
63. Wolfe M. Optimizing Supercompilers for Supercomputers. —Cambridge: MIT Press, 1989.
64. Wolfe M. TINY, a loop restructuring research tool. — Oregon Graduate Institute of Science and Technology, 1990.
65. Xue J. Automatic non-unimodular transformations of loop nests // Parallel Computing, 1994. — Vol. 20, N 5. — P.711–728.
66. Zima H. and Chapman B. Supercompilers for Parallel and Vector Computers. — ACM Press, 1990.

Е. В. Касьянова

ЯЗЫК ПРОГРАММИРОВАНИЯ ZONNON ДЛЯ ПЛАТФОРМЫ .NET*

ВВЕДЕНИЕ

Данная статья описывает состояние пока еще не завершенной работы над языком, получившим название языка Zonnon [1]. Это новый универсальный язык программирования в семействе языков Паскаль, Модула-2 и Оберон. Он сохраняет стремление к простоте, ясному синтаксису и независимости концепций, а также уделяет внимание параллельности и легкости композиции и выражения. Унификация абстракций является стержнем проектирования языка Zonnon, и она отражается в его концептуальной модели, основанной на модулях, объектах, определениях и реализациях. Язык Zonnon содержит такие новые черты, как активность в объектах, основанный на межобъектном взаимодействии диалог, перегрузка операций и обработка исключительных ситуаций. Язык Zonnon специально разрабатывается как платформно-независимый язык.

Сам проект по разработке Zonnon языка возник как продолжение работы его авторов над языком Оберон (Oberon) в проекте 7, который был начат Microsoft Research в 1999 году с целью реализации значительного числа нестандартных языков программирования для платформы .NET [2]. Язык Оберон [3, 4] является хорошо известным преемником языков Паскаль (Pascal) и Модула-2 (Modula-2). Мотивация авторов на продолжение работы по развитию языка Оберон связана со следующими двумя целями [5]:

a) экспериментально исследовать потенциальные возможности платформы .NET в комбинации с новой технологией CCI интеграции компиляторов для проектирования языков;
b) реализовать для платформы .NET язык Zonnon, являющийся эволюцией языка Oberon для .NET.

Поскольку язык Zonnon задуман как дальнейшая эволюция языка Оберон, авторы стремятся сохранить такие важные черты языка Оберон и его преемников, как компактность языка, ясность, недвусмысленность и ортогональность его основных понятий. Вместе с тем, чтобы создать современную альтернативу языку Оберон, авторы внесли в язык ряд изменений, основными из которых являются:

* Работа выполнена при финансовой поддержке Министерства образования РФ (грант № E02-1.0-42) и компании Майкрософт.

- более развитая модульная структура языка;
- продвинутая и одновременно простая и ясная объектная модель;
- концепция активных объектов.

«Общеалгоритмическая» часть языка Zonnon практически полностью повторяет соответствующие части Оберона, поэтому данный язык можно рассматривать как естественную замену Оберону там, где последний традиционно используется для обучения программированию.

Язык Zonnon возник как естественный результат исследований, проводившихся в течение последних нескольких лет в Федеральном Технологическом Институте (ETH) в Цюрихе. Непосредственными предшественниками языка следует считать Active Oberon [6], реализованный как базовый язык ядра операционной системы BlueBottle, а также реализацию языка Oberon для платформы .NET (Oberon.NET). Язык Zonnon, имея ряд общих черт с указанными языками, вместе с тем, существенно отличается от них по ряду принципиальных моментов. Первая реализация Zonnon выполняется для платформы .NET. Кроме того, предполагается интеграция компилятора в систему программирования Visual Studio .NET (в сотрудничестве с компанией Microsoft).

Статья начинается с изложения основных особенностей языка Zonnon (разд. 1). Разд. 2 посвящен вопросам отображения языка Zonnon на платформу .NET. Разд. 3 содержит описание разрабатываемого компилятора с языка Zonnon для платформы .NET. Полный синтаксис языка Zonnon в его состоянии на август 2003 г. приведен в приложении.

1. ЯЗЫК ПРОГРАММИРОВАНИЯ ZONNON

Язык Zonnon — это универсальный императивный язык программирования. Он характеризуется богатой и мощной, но сильно унифицированной объектной моделью, которая поддерживает разнообразные стили программирования, включая обычный стиль алгоритмов и структур данных, стили модульного и объектно-ориентированного программирования, а также модели вычислений, основанные на агентах (actor-based computing models). Основными чертами объектной модели являются

- унифицированная концепция абстракции, так называемое определение (*definition*),
- соответствующее понятие реализации (*implementation*) по умолчанию,
- комбинация обычных объекта и нити (*thread*), называемая активным объектом (*active object*),

• конструкция модуля (*module*).

Нестрого говоря, определения относят к одной категории и унифицируют общие абстракции суперкласса и интерфейса и в комбинации с механизмом статического агрегирования реализаций заменяют концепции иерархии классов и единственного наследования. Активные объекты появляются вместе с интегрированной нитью управления, которая описывает их поведение во время исполнения. Модули — это объекты с управляемой системой жизненным циклом. Кроме того, к языку добавлен оператор блока (*block statement*) с факультативными модификаторами обработки в фигурных скобках и предложением перехвата исключений. Типичными модификаторами обработки являются ACTIVE (исполнять как отдельный поток), EXCLUSIVE (исполнять при взаимном исключении с соответствующей областью действия объекта) и CONCURRENT (все операторы потенциально могут исполняться параллельно). Ниже данные новые языковые понятия будут проиллюстрированы на небольшом наборе простых, но типичных примеров, приведенных авторами языка [3].

1.1. Определения и реализации

Патефон-автомат имеет две “грани” (“facets”): его можно воспринимать либо как хранилище записей, либо как проигрыватель. Соответственно, можно использовать следующие определения:

```
DEFINITION Store;
PROC Clear;
Add (s: Lib.Song);
END Store.
DEFINITION Player;
VAR cur: Lib.Song;
PROC Play (s: Lib.Song);
PROC Stop;
END Player.
```

Предположим, что, кроме того, имеется следующая реализация `Store` по умолчанию:

```
IMPLEMENTATION Store;
VAR rep: Lib.Song;
PROC Clear;
BEGIN rep := NIL
```

```
END Clear;
PROC Add (s: Lib.Song);
BEGIN s.next := rep; rep := s
END Add;
BEGIN Clear
END Store.
```

Тогда можно использовать следующее определение объекта патефона-автомата:

```
OBJECT JukeBox IMPLEMENTS Player, Store;
IMPORT Store; (* aggregate *)
PROCEDURE Play (s: Lib.Song);
IMPLEMENTS Player.Play;
PROCEDURE Stop IMPLEMENTS Player.Stop;
..
END JukeBox.
```

Заметим, что реализация `Store` по умолчанию неявно агрегируется с пространством объектного состояния.

1.2. Активные объекты

В некотором простом террариуме поддерживается следующий вид поведения содержащихся в нем живых тварей. Если температура опускается ниже некоторого определенного минимума, представители всех видов погружаются в спячку, иначе они либо перемещаются случайным образом, либо, если голодны, требуют помощи.

```
OBJECT Creature;
VAR X, Y, temp, hunger, kill: INTEGER;
PROCEDURE NEW (x, y, t: INTEGER);
BEGIN X := x; Y := y; temp := t;
hunger := 0
END;
PROCEDURE SetTemp (dt: INTEGER);
BEGIN { EXCL } temp := temp + dt
END SetTemp;
BEGIN { ACTIVE }
LOOP
AWAIT temp >= minTemp;
```

```
WHILE hunger > minHunger DO
HuntStep(5, kill);
hunger := hunger – kill;
WHILE (kill > 0) & (hunger > 0) DO
HuntStep(7, kill);
hunger := hunger – kill
END;
RandStep(2)
END;
RandStep(4); hunger := hunger + 1
END
END Creature;
```

Заметим, что тело определения объекта последовательно описывает полную историю жизни таких живых тварей. Также заметим, что их поведение еще зависит существенным образом от среды, вызывающей метод `SetTemp`. В частности, любое существо, находящееся в террариуме, может быть заблокировано оператором WAIT до тех пор, пока температура не поднимется выше минимума.

1.3. Модули

Модули — это объекты системного уровня, чей жизненный цикл управляется системой автоматически. В частности, модуль динамически загружается в момент первого вызова. Модули относятся к “статическим” объектам, которые статически могут импортировать другие модули. Хорошим примером системно-ориентированных модулей является менеджер ресурсов. Следующий набросок показывает менеджера окнами с инкапсулированной структурой данных, которая представляет текущую конфигурацию окна в дисплейном пространстве системы. Заметим, что менеджер окнами содержится в пространстве имен, называемом *System*, и что он базируется на другом модуле, называемом `DisplayManager`.

```
MODULE System.WindowManager;
IMPORT System.DisplayManager;
(* static import *)
OBJECT { VALUE } Pos;
VAR X, Y, W, H: INTEGER
END Pos;
DEFINITION Window;
VAR pos: Pos;
```

```
PROCEDURE Draw ();
END Window;
VAR { PRIVATE } W, H: INTEGER;
bot: OBJ { Window };
PROCEDURE Open(this:OBJECT{Window},p:Pos);
BEGIN ...
END Open;
PROCEDURE Change(this:OBJECT{Window},p:Pos);
BEGIN ...
END Change;
BEGIN (* module initialization *) bot:=NIL;
W := System.DisplayManager.Width();
(* delegation *)
H := System.DisplayManager.Height();
END WindowManager.
```

Структурно унифицировано можно представлять систему периода исполнения как ациклическую иерархию модулей. Обычно внизу и наверху иерархии расположены системные модули и модули приложений соответственно.

2. ОТОБРАЖЕНИЕ ЯЗЫКА ZONNON НА ПЛАТФОРМУ .NET

Для любого языка необходимым условием возможности его реализации для .NET является существование отображения его конструкций на общеязыковую исполняющую среду CLR (Common Language Runtime) [2]. В зависимости от парадигмы и модели, представленных языком, это может создавать серьезную проблему. В данном случае императивного языка отображение исполнимой части на механизм исполнения CLR осуществляется непосредственным образом. Что нужно уточнить по существу — это специфицировать отображение определений, реализаций, активных объектов и модулей.

2.1. Отображение определений и реализаций

Существуют различные опции отображения. Если переменные состояний в определениях отображаются в *properties* или *virtual fields* (пусть они существуют), полное пространство состояний может теоретически синтезироваться в реализующем объекте, но с некоторой потерей эффективности. В отличие от этого решение по отображению на C# (канонический язык для

.NET), приведенное ниже, основывается на внутреннем подручном классе, который обеспечивает пространство агрегированных состояний.

```
DEFINITION D;
TYPE e = (a, b);
VAR x: T;
PROCEDURE f (t: T);
PROCEDURE g (): T;
END D;
IMPLEMENTATION D;
VAR y: T;
PROCEDURE f (t: T);
BEGIN x := t; y := t
END f;
END D;
```

отображается на

```
interface D_i {
T x { get; set; }
void f(T t); T g (); }
internal class D_b {
private T x_b;
public enum e = (a, b);
public T x {
get { return x_b };
set { x_b = ... } } }
public class D_c: D_b {
T y;
void f(T t) {
x_b = t; y = t; } }
```

2.2 Отображение активных объектов

Отображение активных объектов в большей степени техническая, а не концептуальная проблема. Очевидно, что средства многопоточности платформы .NET [6] должны служить в этом случае как образы активных конструкций языка Zonnon. Ниже приводится «лобовое» решение для отображения оператора AWAIT. Оно основывается на массовом упоминании ожидаемых объектов в конце каждой критической секции. Авторы надеются, что в дальнейшем им удастся уточнить это решение.

```
BEGIN { ACTIVE } S END
Method void body() { S };
Field Thread thread;
NEW(x)
x.thread = new Thread(
new ThreadStart(body));
x.thread.Start()
AWAIT c
while !c { Monitor.Wait(this); }
BEGIN { EXCL } S END
Monitor.Enter(this); S;
Monitor.PulseAll(this);
Monitor.Exit(this);
```

2.3. Отображение модулей

По существу модули просто отобразить на “статические” классы, которые представляют собой классы только со статическими членами. Ниже приведен набросок описания образа при отображении Zonnon-программ на платформу .NET для приведенного в разд. 1.3. менеджера окнами:

```
namespace System {
namespace WindowManager {
public struct Pos { ... };
public class Window { public Pos pos;
public virtual Draw ();
}
public sealed class WindowManager {
private static int W, H; Window bot;
public static Open (Window this;Pos p)
{ ... };
public static Change(Window this;Pos p)
{ ... }
public static void WindowManager () {
...
W :=
System.DisplayManager.DisplayManager.Width();
...
}}
}
```

3. КОМПИЛЯТОР С ЯЗЫКА ZONNON ДЛЯ .NET

Компилятор с языка Zonnon разработан для платформы .NET и работает наверху нее. Компилятор воспринимает Zonnon исходники (единицы компиляции) и производит обычный ассемблерный код платформы .NET, содержащий MSIL код и метаданные.

Имеются две версии компилятора: компилятор командной строки и компилятор, интегрированный в среду Visual Studio [7]. Компилятор реализован на языке C# с использованием Common Compiler Infrastructure framework (см. ниже), он спроектирован и разработан в Microsoft Research, г. Редмонде [5].

3.1. Common Compiler Infrastructure

Общекомпиляторная инфраструктура CCI (Common Compiler Infrastructure) — это множество ресурсов программирования (C# классы), обеспечивающих поддержку для реализации компиляторов и других языковых инструментов для платформы .NET. Эта поддержка не является полной: не поддержаны некоторые аспекты функциональности компилятора, например, лексический и синтаксический анализ. Но очень важно то, что CCI поддерживает интеграцию в среду MS Visual Studio. Потенциально, возможно достичь полной интеграции компилятора со всеми компонентами VS, такими как редактор, отладчик, менеджер проектов, система онлайновой помощи и т.д.

Среда CCI могла бы рассматриваться как часть всей платформы .NET; пространство имен `Compiler`, содержащее ресурсы CCI, включено в пространство имен `System`. CCI состоит из трех главных частей: промежуточное представление, множество трансфомеров и интеграционная служба.

Промежуточное представление IR (Intermediate Representation) — это богатая иерархия C# классов, представляющих наиболее общие и типичные понятия современных языков программирования. IR иерархия основывается на архитектуре языка C#: ее классы отражают все C# и CLR понятия, такие как класс, метод, оператор, выражение и т.д. Это позволяет разработчику компилятора представлять напрямую подобные понятия его языка. В случае, когда язык имеет понятия или конструкции, которые не представлены множеством IR классов, есть возможность расширять исходную IR иерархию добавлением новых IR классов. При этом должны быть добавлены также соответствующие трансформации — либо как расширения стандартных “визитёров” (см. ниже), либо в виде полностью нового визитёра.

Трансфомеры (“визитёры”) представляют собой множество основанных классов, выполняющих последовательные преобразования из IR в ассемблерный код платформы .NET. Имеются пять стандартных визитёров, предопределенных в CCI: наблюдатель (Looker), описатель (Declarer), решатель (Resolver), проверяющий (Checker), и нормализатор (Normalizer). Все визитёры обходят IR, выполняя преобразования различных видов. Визитёр наблюдатель (в компании с визитёром-описателем) заменяет вершины Identifier на числа/локалы, к которым они сводятся. Визитёр-решатель разрешает перегрузки операций и выводит типы результатов выражений. Проверяющий визитёр выявляет семантические ошибки и пытается устранить их, а визитёр-нормализатор готовит IR для выстраивания в MSIL и метаданные.

Все визитёры реализованы как классы, наследуемые от класса StandardVisitor из CCI. Допускается как расширение функциональности некоторого визитёра путем добавления методов, обрабатывающих специфические языковые конструкции, так и создание некоторого нового визитёра. CCI требует, чтобы все визитёры, используемые в компиляторе, наследовались (прямо или косвенно) от класса StandardVisitor.

Интеграционная служба представляет собой разнообразные классы и методы, обеспечивающие интеграцию в среду Visual Studio. Указанные классы инкапсулируют специфические структуры данных и функциональность, необходимые для редактирования, отладки, фоновой компиляции и т.д.

3.2. Архитектура компилятора с языка Zonnon

Концептуально организация компилятора вполне традиционна: сканер (Scanner) осуществляет преобразование исходного текста в лексическую свертку, которая воспринимается парсером (Parser). Парсер осуществляет синтаксический анализ и строит абстрактное синтаксическое дерево (AST), используя для входной единицы компиляции классы CCI IR. Каждая вершина AST является экземпляром класса IR. “Семантическая” часть компилятора состоит из серии последовательных преобразований AST, построенного парсером. Результатом таких трансформаций является ассемблерный код платформы .NET, который и является результатом работы компилятора.

Однако с архитектурной точки зрения компилятор с языка Zonnon отличается от большинства “традиционных” компиляторов. Вместо использования технологии “черного ящика”, когда все алгоритмы компилятора и структуры данных инкапсулируются в компиляторе и не видны извне, ком-

пилятор с языка Zonnon, по существу, представляет собой открытый набор ресурсов. В частности, такие структуры данных, как лексическая свертка и дерево AST доступны (через специальный интерфейс) извне компилятора. Аналогичное свойство имеет место для компонентов компилятора: например, возможно вызвать сканер с тем, чтобы извлечь лексемы из конкретной части входа, а затем вызвать парсер, чтобы построить поддерево для этой части входа.

Такая архитектура поддерживается схемой CCI и обеспечивает глубокую и естественную интеграцию компилятора в среду Visual Studio. Для того чтобы поддержать интеграцию, CCI содержит прототипные классы для сканера и парсера. Фактические реализации компонентов сканера и парсера компилятора с языка Zonnon представляют собой классы, наследуемые от этих прототипных классов.

Компилятор расширяет множество вершин IR путем добавления большого числа конкретных типов вершин для понятий Module, Definition, Implementation, Object и для некоторых других конструкций языка Zonnon, которые не имеют прямых прототипов среди CCI-вершин. Дополнительные вершины обрабатываются расширенным визитёром-наблюдателем, который наследует стандарный класс Looker из CCI. Результат работы расширенного наблюдателя семантически эквивалентен дереву AST, содержащему только предопределенные типы CCI. Следовательно, расширенный наблюдатель реализует отображение, описанное в разд. 2.

ЗАКЛЮЧЕНИЕ

В статье представлен язык программирования Zonnon, работа над которым ведется в Цюриховском институте информатики. Разрабатываемый язык задуман как дальнейшая эволюция хорошо известного языка Оберон, являющегося преемником языков Паскаль и Модула-2. Развивая язык Оберон, исходя из современных потребностей в программировании, авторы стремятся сохранить такие важные черты Оберона и его предшественников, как компактность языка, ясность, недвусмысленность и ортогональность его основных понятий. Поэтому можно ожидать, что язык будет востребован теми учебными заведениями, которые в настоящее время используют Паскаль в качестве языка начального обучения программированию, но имеют желание перейти к более современному курсу программирования, охватывающему концепции языков программирования нового поколения,

таких как Java и C#, но осуществить этот переход плавно, без резкого изменения сложившегося стиля преподавания программирования.

СПИСОК ЛИТЕРАТУРЫ

1. Gutknecht J., Zueff E. Zonnon Language Report. — Zurich: Institute of Computer Systems ETH Zentrum, 2003.
2. Уоткинз Д., Хаммонд М., Эйбрамз Б. Программирование на платформе .NET. — М.: Вильямс, 2003.
3. Wirth N. The programming language Oberon // Software — Practice and Experience. — 1988. — Vol. 18, N 6. — P. 671–690.
4. Wirth N. From Modula to Oberon // Software — Practice and Experience. — 1988. — Vol. 18, N 6. — P. 661– 670.
5. Gutknecht J., Zueff E. Zonnon Language Experiment, or How to Implement a Non-Conventional Object Model for .NET // OOPSLA'02. — Seattle, Washington, 2002.
6. Просиз Дж. Программирование для .NET. — М.: Русская Редакция, 2003.
7. Джонсон Б., Скибо К., Янг М. Основы Microsoft Visual Studio .NET 2003. — М.: Русская Редакция, 2003.

ПРИЛОЖЕНИЕ

Ниже для описания синтаксиса языка Zennon используется Расширенный Бекуса—Наура Формализм (Extended Backus—Naur Formalism, EBNF), который характеризуется следующими свойствами:

- альтернативы разделяются символом |;
- скобки [и] обозначают факультативность выражения в скобках;
- скобки { и } обозначают повторение (возможно 0 раз);
- скобки (и) используются для формирования групп элементов;
- нетерминальные символы начинаются с прописной буквы (например, Statement);
- терминальные символы либо начинаются со строчной буквы (например, letter*)*, либо записывается целиком строчными буквами (например, BEGIN), или представляются в виде строк (например, ":=");
- комментарии начинаются с символа // и продолжаются до конца строки.

```
// Синтаксис языка Zonnon в EBNF
// Версия от 8 августа 2003
// 1. Программа и программные единицы
CompilationUnit = { ProgramUnit "." }.
ProgramUnit = ( Module | Definition | Implementation | Object ).
// 2. Модули
Module = MODULE ModuleName [ ImplementationClause ] ";"
[ ImportDeclaration ]
ModuleDeclarations
( BlockStatement | END ) SimpleName.
ModuleDeclarations = { SimpleDeclaration | NestedUnit ";" }
{ ProcedureDeclaration | OperatorDeclaration }
{ ActivityDeclaration }.
NestedUnit = ( Definition | Implementation | Object ).
ImplementationClause = IMPLEMENTS DefinitionName { "," DefinitionName }.
ImportDeclaration = IMPORT Import { "," Import } ";".
Import = ImportedName [ AS ident ].
ImportedName = ( ModuleName | DefinitionName | ImplementationName | Namespace-
    Name ).
// 3. Определения
Definition = DEFINITION DefinitionName [ RefinementClause ] ";"
[ ImportDeclaration ]
DefinitionDeclarations
END SimpleName.
RefinementClause = REFINES DefinitionName.
DefinitionDeclarations = { SimpleDeclaration } { ProcedureHeading | ActivitySignature
    ";" }.
ActivitySignature =
ACTIVITY ActivityName[ TYPE "{" ProtocolEBNF "}" [ EnumType] END Simple-
    Name] ";".
ProtocolEBNF = Спецификация протокола в EBNF на основе алфавита лексем.
// 4. Реализации
Implementation = IMPLEMENTATION ImplementationName ";"
[ ImportDeclaration ]
Declarations
( BlockStatement | END ) SimpleName.
// 5. Объекты
Object = OBJECT [ ObjModifier ] ObjectName [ FormalParameters ] [ Implementation-
    Clause ] ";"
[ ImportDeclaration ]
Declarations
{ ActivityDeclaration }
( BlockStatement | END ) SimpleName.
```

```
ObjModifier = "{" ident "}". // VALUE or REF; VALUE by default
ActivityDeclaration = ACTIVITY ActivityName [ ImplementationClause ] ";"
Declarations ( BlockStatement | END SimpleName ).
// 6. Объявления
Declarations = { SimpleDeclaration } { ProcedureDeclaration }.
SimpleDeclaration = ( CONST [DeclModifier] { ConstantDeclaration ";" }
| TYPE [DeclModifier] { TypeDeclaration ";" }
| VAR [DeclModifier] { VariableDeclaration ";" }
).
DeclModifier = "{" ident "}". // PUBLIC or PRIVATE or IMMUTABLE
ConstantDeclaration = ident "=" ConstExpression.
ConstExpression = Expression.
TypeDeclaration = ident "=" Type.
VariableDeclaration = IdentList ":" Type.
// 7. Типы
Type = ( TypeName [ Width ] | EnumType | ArrayType | ProcedureType | InterfaceType ).
Width = "{" ConstExpression "}".
ArrayType = ARRAY Length { "," Length } OF Type.
Length = ( Expression | "*" ).
EnumType = "(" IdentList ")".
ProcedureType = PROCEDURE [ ProcedureTypeFormals ].
ProcedureTypeFormals = "(" [ PTFSection { ";" PTFSection } ] ")" [ ":" FormalType ].
PTFSection = [ VAR ] FormalType { "," FormalType }.
FormalType = { ARRAY OF } ( TypeName | InterfaceType ).
InterfaceType = OBJECT [ PostulatedInterface ].
PostulatedInterface = "{" DefinitionName { "," DefinitionName } "}".
// 8. Процедуры и знаки операций
ProcedureDeclaration = ProcedureHeading [ ImplementationClause ] ";" [ ProcedureBody
    ";" ].
ProcedureHeading = PROCEDURE [ ProcModifiers ] ProcedureName [ FormalParameters
    ].
ProcModifiers = "{" ident { "," ident } "}". // PRIVATE, PUBLIC, SEALED
ProcedureBody = Declarations BlockStatement SimpleName.
FormalParameters = "(" [ FPSection { ";" FPSection } ] ")" [ ":" FormalType ].
FPSection = [ VAR ] ident { "," ident } ":" FormalType.
OperatorDeclaration = OPERATOR [ OpModifiers ] OpSymbol [ FormalParameters ] ";"
OperatorBody ";".
OperatorBody = Declarations BlockStatement OpSymbol.
OpModifiers = "{" ident { "," ident } [ "," Priority ] "}"
.| "{" Priority "}".
Priority = ConstExpression.
OpSymbol = string. // Строка из 1, 2 или 3 символов из
// ограниченного множества возможных символов
```

```
// 9. Операторы
StatementSequence = Statement { ";" Statement }.
Statement = [ Assignment
| ProcedureCall
| IfStatement
| CaseStatement
| WhileStatement
| RepeatStatement
| LoopStatement
| ForStatement
| AWAIT Expression
| EXIT
| RETURN [ Expression ]
| BlockStatement
| Send
| Receive
].
Assignment = Designator ":=" Expression.
ProcedureCall = Designator.
IfStatement = IF Expression THEN StatementSequence
{ ELSIF Expression THEN StatementSequence }
[ ELSE StatementSequence ]
END.
CaseStatement = CASE Expression OF
Case { "|" Case }
[ ELSE StatementSequence ]
END.
Case = [ CaseLabel { "," CaseLabel } ":" StatementSequence ].
CaseLabel = ConstExpression [ ".." ConstExpression ].
WhileStatement = WHILE Expression DO StatementSequence END.
RepeatStatement = REPEAT StatementSequence UNTIL Expression.
LoopStatement = LOOP StatementSequence END.
ForStatement = FOR ident ":=" Expression TO Expression [ BY ConstExpression ]
DO StatementSequence END.
BlockStatement = BEGIN [ BlockModifiers ]
StatementSequence
{ ExceptionHandler }
[ CommonExceptionHandler ]
END.
BlockModifiers = "{" ident { "," ident } "}". // LOCKED, CONCURRENT, BARRIER
ExceptionHandler = ON ExceptionName { "," ExceptionName } DO StatementSequence.
CommonExceptionHandler = ON EXCEPTION DO StatementSequence.
Send = [Designator] "!" Expression.
```

```
Receive = [Designator] "?" Designator.
// 10. Выражения
Expression = SimpleExpression
[ ( "=" | "#" | "<" | "<=" | ">" | ">=" | IN ) SimpleExpression ]
| Designator IMPLEMENTS DefinitionName
| Designator IS TypeName.
SimpleExpression = [ "+"|"" ] Term { ( "+" | "" | OR ) Term }.
Term = Factor { ( "*" | "/" | DIV | MOD | "&" ) Factor }.
Factor = number
| CharConstant
| string
| NIL
| Set
| Designator
| NEW TypeName [ "(" ActualParameters ")" ]
| NEW ActivityInstanceName
| "(" Expression ")"
| "~" Factor.
Set = "{" [ SetElement { "," SetElement } ] "}".
SetElement = Expression [ ".." Expression ].
Designator = Instance
| Designator "^" // Dereference
| Designator "[" Expression { "," Expression } "]"// Array element
| Designator "(" [ ActualParameters ] ")" // Function call
| Designator "." MemberName // Member selector
Instance = ( SELF | InstanceName | DefinitionName "(" InstanceName ")" ).
ActualParameters = Actual { "," Actual }.
Actual = Expression [ "{" [ VAR ] FormalType "}" ]. // Аргумент с сигнатурой типа
// 11. Константы
number = (integer | real) [ "{" Width "}" ].
integer = digit { digit } | digit { HexDigit } "H".
real = digit { digit } "." { digit } [ ScaleFactor ].
ScaleFactor = "E" ["+" | "" ] digit { digit }.
HexDigit = digit | "A" | "B" | "C" | "D" | "E" | "F".
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
CharConstant = "" character "" | "" character "" | digit { HexDigit } "X".
string = "" { character } "" | "" { character } "".
character = letter | digit | Other.
Other = // Любой символ алфавита, кроме тех, которые используются …
// 12. Идентификаторы и имена
ident = ( letter | "_" ) { letter | digit | "_" }.
letter = "A" | ... | "Z" | "a" | ... | "z" | любая другая "культурно-определенная" буква
IdentList = ident { "," ident }.
```

```
QualIdent = { ident "." } ident.
DefinitionName = QualIdent.
ModuleName = QualIdent.
NamespaceName = QualIdent.
ImplementationName = QualIdent.
ObjectName = QualIdent.
TypeName = QualIdent.
ExceptionName = QualIdent.
InstanceName = QualIdent.
ActivityInstanceName = QualIdent.
ProcedureName = ident.
ActivityName = ident.
MemberName = ( ident | OpSymbol ).
SimpleName = ident.
```

Ю. В. Малинина

ЭЛЕКТРОННАЯ СРЕДА КОЛЛЕКТИВНОГО НАКОПЛЕНИЯ И КАТАЛОГИЗАЦИИ ИНФОРМАЦИИ ПО ПРЕОБРАЗОВАНИЯМ ПРОГРАММ*

ВВЕДЕНИЕ

Передача знаний происходит во время общения между людьми, направленного на получение необходимых знаний для решения задач или принятия решений. Коммуникации могут быть персональными и групповыми, непосредственными или заочными. Персональные коммуникации (общение) используются в повседневной жизни, например, тогда, когда человек обращается за советом или консультацией к коллеге. Для обеспечения эффективности групповых коммуникаций применяются различные мероприятия, организующие процесс общения в группах. К таким мероприятиям можно отнести привычные совещания, семинары, конференции, съезды и т.д. Заочные коммуникации обычно происходят, например, посредством бумажных или электронных документов и писем.

Между учеными до сих пор нет единого мнения о том, что такое теория коммуникации и что должно быть предметом ее исследования. Профессор Роберт Т. Крейг [2] считает, что существование разных трактовок понятия «коммуникация» связано с тем, что исследованиями в этой области занимаются специалисты разных профессий, каждый из которых рассматривает коммуникации применительно к своему роду деятельности и зачастую не интересуется работами коллег из других сфер. Однако уже не вызывает сомнения то, что Интернет и сетевые технологии могут существенно влиять на процесс коммуникаций. Особое внимание привлекают различные способы асинхронного взаимодействия, которые снимают трудности, связанные с организацией групповых и заочных коммуникаций. Если раньше они были ограничены текстовой формой электронных писем и бюллетеней, то теперь эта форма взаимодействия стала более гибкой.

Если обратится к творческому наследию Герберта Маршалла Маклюэна, то в своих монографиях он утверждал, что западная цивилизация дос-

* Работа выполнена при финансовой поддержке Научной программы «Университеты России» (грант № УР.04.01.027) и Министерства образования РФ (грант № Е02-1.0-42).

тигла «водораздела» в XX веке, имеющего не меньшее значение, чем эпоха Ренессанса. Качественные сдвиги в истории человечества, согласно Маклюэну [3], связаны с появлением новых технических средств коммуникации. Отсюда и следует его знаменитое высказывание о том, что "средство и есть содержание" (“The Medium is the Message” — сообщением, передаваемым средством общения, является само это средство). Он говорит, что попытки выполнять сегодняшние обязанности с помощью вчерашних орудий — вчерашних концепций, где информация скудна, но упорядочена и построена по отдельным классифицированным планам, темам и графикам, уже не отвечают современным требованиям. “Технологии — это не просто изобретения, которые используют люди, это средства, с помощью которых люди создаются заново... Будущее — адрес нашего проживания".

Согласно Нельсону [4], одному из пионеров гипертекстов, реальная структура документов и источников игнорируется существующими сегодня форматами, имитирующими бумажные документы. Пересечение и параллелизм документов, ключевыми примерами которых могут служить аннотации и взаимные связи, различные схемы цитирования и доступность оригинала цитируемого текста, практически не используются в повседневной работе.

Сегодня можно заметить, что компьютерные науки ставят в центр своих изысканий два традиционных конструкта — имитацию иерархии и имитацию бумаги. Существует популярный миф, что «структура» всегда означает иерархию. Существует также популярная концепция, что электронный документ должен имитировать документ бумажный.

Зачастую информационная структура не может быть верно представлена иерархией. Такие взаимосвязи, как параллелизм, перекрестные связи, взаимное проникновение и взаимное присутствие (присутствие одного элемента в нескольких местах), теряются при построении иерархии, когда делается выбор в пользу только одного типа отношений. Например, в представлении файловых систем вы можете выбрать только один тип иерархии и расположить объекты по шкале времени или важности в ущерб всем остальным типам отношений. Эти отношения становятся основными и определяют все последующие соглашения — псевдонимы, сокращения и базы данных.

Как следствие, суперсовременные системы работы с документами не соответствуют тому гигантскому потенциалу, которым обладает компьютер, не используют и малой толики из возможностей, которые открываются в случае, если документ представлен в форме, читаемой машиной. Вот и вы-

ходит, что, в конечном счете, компьютер на столе служит не более чем удобной пишущей машинкой и компактным, безопасным архивом.

Однако если прислушаться к Нельсону и покинуть привычное для нас поле традиционных структур, перед нами открываются многочисленные альтернативные возможности. Рассмотрим некоторые из них.

НОВЫЕ ВОЗМОЖНОСТИ

Тот факт, что научная деятельность и творчество уже не возможны без существенных изменений в способах коммуникации и способах размышления, отмечался уже в середине прошлого столетия в работах основателей кибернетики. Еще Ваннавер Буш описывал новый способ мышления при помощи машинных средств обработки и передачи информации [1]. Валентин Турчин рассматривает творчество как метасистемный переход и также полагает, что использование машин позволит перевести мыслительную деятельность на более высокий уровень [5]. Новые цифровые технологии могут освободить наш мозг от механических аспектов мышления для решения творческих задач.

WikiWiki — коллективный гипертекст

Не секрет, что процесс мышления обладает определенной спецификой: идеи развиваются на разных уровнях, в разных направлениях одновременно и взаимосвязано. Записывать такую систему идей и потом читать ее достаточно трудно. Традиционные плоские текстовые файлы связывают нас необходимостью писать и читать параграфы текста, главным образом, в линейной форме. В традиционной книжной форме представления текста присутствуют элементы нелинейности: комментарии в скобках, комментарии в сносках, ссылки на предыдущие или последующие разделы, ссылки на литературу, фрагменты текста, выделенные другим шрифтом, которые позволяют автору неявно сказать: «Это не входит в основной текст, но имеет к нему отношение; если Вам интересно, можете посмотреть». Кроме того, существует много стилистических оборотов, позволяющих представить нелинейный по существу ход рассуждений в последовательной линейной форме. Однако этих форм нелинейности не всегда бывает достаточно, не всегда они удобны при просмотре текста, особенно на компьютере.

Известно, что существующие тексты и без компьютерных сетей образуют единое пространство [6]. Но при этом система адресации имеет достаточно запутанный характер, поскольку хранится частично в книжной, а час-

тично — в человеческой памяти. В действительности библиотечные и информационные науки в основном и состоят из исследования возможностей ссылок. Любой, кто проводил исследования, понимает, что значительную часть работы составляет получение нужной литературы, просмотр ссылок на литературу, просмотр терминов в справочниках или глоссариях, проверка таблиц или рисунков, анализ оглавлений или перечислений, составление собственных пометок. Даже при простом чтении постоянно встречаются ссылки на другие разделы или главы, подстрочные комментарии, литературу, примеры, таблицы, рисунки, приложения. Часто некоторые главы пропускаются читателем, которого не интересуют технические подробности. Однако перед традиционной литературой стоят определенные проблемы.

- Большинство ссылок не помогают вернуться назад: читателю бывает очень сложно найти те книги или статьи, которые ссылаются на ту статью, которую он читает. И электронные варианты публикаций не являются исключением, а с завидным упорством повторяют этот подход.
- Читатель должен размещать свои пометки либо на полях книги или статьи, либо помещать их на отдельном листе. Подобный подход обычно применяется и при работе с файлами.
- Наконец, следование по ссылкам в бумажных документах (или их электронных аналогах) требует постоянной затраты сил, внимания, дает задержки, даже если читатель работает в хорошо оборудованной библиотеке.

Обратим свое внимание на ставший уже привычным при нынешнем развитии Интернета – гипертекст и попробуем посмотреть на него с другой стороны.

Сама концепция гипертекста достаточно проста. Однако несмотря на свою простоту, эта идея дала толчок многим интересным идеям и разработкам.

Фактически, гипертекст — это множество документов, связанных между собой ссылками. На первый взгляд, основная суть гипертекста состоит в том, чтобы предоставлять быстрый и удобный интерфейс к текстовым фрагментам. Но такое описание ничего не говорит об его реальной роли в развитии коммуникаций. Однако, если рассмотреть гипертекст как специальную, поддерживаемую компьютером среду для мышления и общения, можно сделать ряд интересных наблюдений.

Наиболее прогрессивная и радикальная модель коллективного гипертекста — WikiWiki. Тексты WikiWiki связаны между собой при помощи самой простой системы адресации. Само слово — название ресурса — и является его адресом в базе данных. Для того чтобы сделать ссылку на ресурс, достаточно просто упомянуть это слово в тексте.

Гипертекст (WikiWiki) позволяет автору делать подобные ссылки, а читателю самому выбирать, по каким связям он будет идти при просмотре данного текста. С этой точки зрения, WikiWiki упрощает работу мыслителя и автора: не требуется постоянно решать — вводить ли данную мысль в основной поток изложения или нет. В этом смысле «ссылочность» является основой мощности гипертекстов: именно ссылки, поддерживаемые компьютером, расширяют текст за пределы одномерного потока.

В то же время, если гипертекст используется как инструмент для мышления, написания, проектирования, может возникнуть естественное соответствие между объектами окружающего мира и узлами в базе данных гипертекста. Пользуясь достоинствами такого объектно-ориентированного аспекта, при помощи гипертекста можно построить гибкую сеть, которая будет моделировать некоторую проблему (или ее решение). В таких приложениях связи могут быть даже менее важными, чем узлы: связи формируют смысловые группы, которые собирают узлы вместе, но основной акцент делается именно на узлы.

WikiWiki позволяет легко использовать в повседневной деятельности все преимущества, предоставляемые гипертекстами.

- *Простота следования по ссылкам:* поддержка компьютером при следовании по ссылкам позволяет одинаково легко следовать как вперед по ссылкам, так и возвращаться назад.

- *Простота создания новых ссылок:* пользователь может развивать свою сеть или просто комментировать чей-то документ.

- *Структуризация информации:* к неструктурированной информации можно применять как иерархический способ организации, так и неиерархический; более того, можно на одном и том же материале организовать несколько разных иерархий.

- *Глобальный взгляд:* специальные системы просмотра могут обеспечить глобальный взгляд на документ как на сеть узлов, что существенно для очень больших или сложных документов.

- *Текстовые узлы* могут быть собраны вместе различными способами, давая возможность одному и тому же документу выполнять различные функции.

- *Модульность информации:* так как на один и тот же текстовый сегмент можно ссылаться из нескольких мест, мысли могут быть выражены с меньшими перекрытиями и дублированием.

- *Связность информации:* ссылки становятся неотъемлемой частью текста, и если какая-то часть текста переносится в другое место, даже в другой документ, информационные ссылки продолжают давать прямой доступ к данному фрагменту текста.

- *Взаимодействие с использованием документа:* то, что пользователь сам выбирает путь, по которому он просматривает гипертекст, делает пользователя активным участником процесса соотнесения гипертекстового документа к задаче, стоящей перед пользователем.

- *Совместная работа:* возможность совместной работы нескольких авторов над одним документом, что дает принципиально новые возможности.

Конечно, при увеличении объема и сложности информации, представляемой в виде гипертекста, зачастую читателю все труднее становится понимать, в каком месте сети он сейчас находится и как ему перейти в другое нужное место гипертекста. Эта проблема возникает из-за того, что в гипертексте, в отличие от линейного текста, у пользователя гораздо больше возможных путей просмотра, больше степеней свободы.

Существуют два основных технических решения, помогающих решать эту проблему: подсистема графического просмотра гипертекстовой сети и механизм поиска по запросу. Подсистема графического просмотра сети — это подсистема, позволяющая в явном виде представить структуру гипертекста в виде сетевого графа и показать положение пользователя в этой сети. Естественно, такие подсистемы требуют очень высокой разрешающей способности компьютерных дисплеев. Кроме того, для того чтобы разместить узлы и связи в двух- или трехмерном пространстве, придать им полезные визуальные особенности (цвет, форму, размер, фактуру), придерживаться полезных соглашений (не размещать два узла слишком близко друг от друга и т.п.), разработчики таких подсистем должны решить очень сложную программистскую задачу. Но даже и такое сложное программирование все равно оказывается стоящим перед тем фактом, что для сложного ин-

формационного пространства просто не существует естественной топологии.

Другое решение этой проблемы — применить типовую технику поиска по запросу из баз данных. Запрос на поиск обычно организуется в виде некоторой логической комбинации поиска по ключевому слову, по подстроке, по свойствам узлов или связей.

Создание документа в WikiWiki

Работа авторов документов обычно рассматривается как работа на уровне слов и предложений. Ясно, что текстовый редактор — это хороший инструмент для авторов. Если же рассматривать работу по написанию текстов как работу по структуризации идей, упорядочиванию представлений, использованию понятий, текстового редактора становится недостаточно. Очень немногие авторы просто садятся и пишут законченный текст. В каком-то смысле авторы — это проектировщики текстовых документов. И единицей такого уровня авторской работы является мысль или понятие, и работа такого уровня может быть легко поддержана при помощи гипертекстов. По мере того как автор приходит к новым мыслям, он может развивать их в виде узлов и затем привязывать эти мысли к единому целому или делать их изолированными. Таким образом, гипертексты помогают автору двигаться от слабо структурированной сети мыслей к законченному документу, годному для печати.

WikiWiki вводит иной подход создания Интернет страниц. Обычно требуется, чтобы сначала была создана страница, а уже затем на эту страницу была бы сделана ссылка. В WikiWiki в таком предварительном создании нет необходимости. Всякое новое определение сначала вводится в тексте, а потом уже может быть разъяснено. Мы можем ввести в тексте новые термины, например, так называемое WikiWord-слово. После сохранения текста мы получаем страничку, на которой после слова-определения WikiWord появилась ссылка со знаком вопроса, сигнализирующая о том, что определение этого термина еще не задано. Нажав на эту ссылку, мы создадим новую страницу, которая будет содержать определение понятия WikiWord.

Преимущество WikiWiki технологии заключается в том, что она позволяет реализовать коллективную работу с гипертекстовыми документами с использованием простого языка разметки и обычного браузера. Например, должно быть выработано общее определение и представление о понятии «преобразование программы». В WikiWiki нельзя завести несколько объектов ПреобразованиеПрограммы. С другой стороны, к правке понятия Пре-

образованиеПрограммы имеют доступ все участники, и таким образом реализуется возможность хранить несколько точек зрения на некоторое понятие в одном ресурсе, которая особенно важна в случаях, когда в рамках совместной работы формируются и обсуждаются термины и определения.

На сегодня существует много различных реализаций WikiWiki; их объединяет то, что WikiWiki-страницы имеют два представления. Одно — это вид, который требуется для вывода (обычно это HTML), а второе, доступное для пользователей, пишется на упрощенном языке разметки, незначительно отличающемся от естественного языка, синтаксис и стиль которого могут варьироваться в зависимости от конкретной реализации. Система WikiWiki дает удобные возможности при создании сложных источников информации. Линейный (не гипертекстовый) документ нормально может читаться только единственным способом — каким он организован. У нелинейного же текста есть существенное достоинство: такой текст можно организовать различными способами, в зависимости от точек зрения.

Идея WikiWiki хорошо воспринята целым рядом технологических сообществ, но подобные среды оставались атрибутами этих сообществ, не выходя вовне, размеры баз данных в них не превышали нескольких сотен страниц. Настоящий прорыв произошел в 2001 г., когда Лари Сандр дал старт проекту WikiWikipedia. На октябрь 2003 г. число статей в WikiWikipedia превышает 150 тыс.

Ссылки по теме:

- `http://c2.com/cgi/WikiWiki` — самый первый WikiWiki. — WikiWiki история развивалась как техническая поддержка хранилища образцов и моделей (Patterns).
- `http://www.WikiWikipedia.org/` — сайт-энциклопедия в стиле WikiWiki. (http://ru.WikiWikipedia.org/)

К несомненным достоинствам WikiWiki относится то, что этот продукт можно использовать и в индивидуальной, и в коллективной практике. Для индивидуального использования достаточно на личном компьютере установить Интернет-сервер и какой-нибудь из клонов WikiWiki (возможно понадобится установка Perl, php, java – в зависимости от выбранного клона), и все это будет быстро и хорошо работать. WikiWiki является хорошим инструментом для хранения записок по разным темам.

Обычно WikiWiki поддерживает следующие возможности редактирования.

- Выделение текста.
- Создание списочных структур.
- Простое форматирование текста.
- Автоматическое задание ссылки для слов определенного формата.
- Явное указание ссылок или их отмена.
- Использование некоторых HTML тэгов.

Интересного эффекта можно добиться интеграцией WikiWiki с коллекцией. В результате этого симбиоза к привычной для всякой WikiWiki возможности создания и редактирования гипертекста добавилась возможность напрямую ссылаться на нередактируемые тексты, собранные в коллекции. Ссылки на авторов коллекции ведут к их текстам. К авторским текстам добавилась возможность «кликнуть» имя автора текста и получить страницы WikiWiki, на которых происходит обсуждение и использование этого текста (своего рода заметки на полях электронных текстов в стиле WikiWiki).

Основные возможности, предоставляемые WikiWiki.

- Относительная простота «входа» в систему.
- Возможность редактирования любой страницы системы любым пользователем.
- Возможность многопользовательской работы над документами.
- При многопользовательской работе не надо строго формализовать работу группы.
- Контроль версий документов.
- Возможность проставления ссылок на несуществующие страницы, а также возможность проставления ссылок по названию страниц, а не по адресу.
- Упрощенная разметка текста: это не WYSIWYG, но все же и не HTML. Впрочем, это можно отнести и к минусам, зависит от того, как вы привыкли редактировать.

WikiWiki как хранилище групповой памяти

Интерес к WikiWiki связан с поиском средств, которые бы поддерживали коллективную научную деятельность. Идея построения направленного графа из неформальных текстовых элементов близка идее семантической сети из области искусственного интеллекта. Семантическая сеть — это схема представления знаний, состоящая из направленного графа, в котором понятия представляются как узлы, а отношения между этими понятиями — как связи между узлами.

Семантическую сеть отличает то, что понятия индексируются посредством их семантического содержания, а не каким-либо внешним (возможно, алфавитным) порядком. Одно из преимуществ семантических сетей перед другими схемами представления знаний заключается в том, что использование их естественно, т.к. близкие понятия стремятся связаться в единый пучок сети. Соответственно, легко распознать неполные или неадекватные понятия, так как смысловой контекст определяется именно связями с соседними понятиями.

Здесь можно увидеть прямую аналогию с гипертекстами, и в частности с WikiWiki технологией. Отличие состоит в том, что инженер знаний стремится построить такие представления, которые могли бы автоматически интерпретироваться, в то время как цель автора гипертекста — собрать взаимосвязанную систему мыслей, не претендуя на машинную интерпретацию.

ЗАКЛЮЧЕНИЕ

Итак, WikiWiki — не случайное событие, а результат закономерного эволюционного процесса развития гипертекстов. Использование WikiWiki в качестве инструмента для организации среды построения каталога по преобразованиям программ, на наш взгляд, позволит найти дополнительные взаимосвязи в области преобразований программ и облегчить повседневную работу специалистов в этой области.

WikiWiki позволяет организовать в сети для всех участников коллективной деятельности общее равнодоступное пространство, где они могут размещать свои мысли в форме письменных высказываний, связывать эти высказывания между собой, редактировать и корректировать свои и чужие высказывания. Такую электронную среду можно рассматривать как аналог доски, на которой может писать и рисовать каждый. Электронная среда имеет очевидное преимущество, поскольку неограниченные размеры позволяют хранить всю историю взаимодействия и все разнообразие связей. Публикуемые в сети документы наряду с текстом могут содержать ссылки на графику, диаграммы, звуковые файлы и другие связанные ресурсы. Общая деятельность может быть представлена в форме, облегчающей индивидуальное восприятие.

СПИСОК ЛИТЕРАТУРЫ

1. Bush V. As we may think // The Atlantic Monthly. —1945. — Vol. 176(1). — P.101–108 — http://www.csi.uottawa.ca/~dduchier/misc/vbush/awmt.html
2. Craig R.T. Communication Theory as a Field // Communication Theory. —1999. — N 9 — P. 217–242 — http://www.comm.cornell.edu/comm680/craig.pdf.
3. McLuhan M. The Medium is the Message. — NY, 1967. — http://uic.nnov.ru/pustyn/lib/maclu.ru.html
4. Nelson T. Structure, traditions and possibility // Proc. of the 14th ACM Conf. on Hypertext and hypermedia, Nottingham, UK, August 26–30, 2003. — ACM Press, 2003. — P. 1–1
5. Турчин В. Феномен науки. Кибернетический подход к эволюции — М: ЭТС, 2000. — 368 с. — http://www.refal.ru/turchin/phenomenon/index.htm
6. Фуко М. Археология знания. — Киев: Ника-центр, 1996. — 208с. — http://yanko.lib.ru/books/cultur/foucalt_larcheologie_du_savoir_ru.htm

В. А. Маркин, С. А. Маркина

СИСТЕМА ДЛЯ БЫСТРОГО ПРОТОТИПИРОВАНИЯ РАСПАРАЛЛЕЛИВАЮЩЕГО КОМПИЛЯТОРА ПРОГРЕСС-2. ЯДРО СИСТЕМЫ. СЦЕНАРИЙ СИСТЕМЫ[*]

ВВЕДЕНИЕ

Данная статья является продолжением статьи [1], описывающей систему для быстрого создания прототипа распараллеливающего компилятора. Система создается как инструмент для манипулирования программами; что означает пошаговое преобразование программ — начиная с исходного текста программы на одном из языков высокого уровня, переводим программу в одно из внутренних представлений и далее пропускаем программу через различные проходы-блоки. В конце концов, мы можем получить текстовое представление распараллеленной программы либо исполняемый код для запуска программы на одной из целевых архитектур. Теоретические размышления и предпосылки создания подобной системы можно найти в работах [2,3] по проекту ПРОГРЕСС, в рамках которого и начиналась работа над описанной далее системой.

Центральным понятием работы системы является СЦЕНАРИЙ прототипа создаваемого компилятора. В сценарии определяется множество и порядок исполнения функциональных и инструментальных компонент системы. Компонентом системы может быть либо front-end с одного из языков высокого уровня, обход дерева внутреннего представления системы либо анализ свойств программ. Компонент может быть интерактивным, ориентированным на взаимодействие с пользователем, иметь внешний графический интерфейс. Такие компоненты, например, могут служить для визуализации одного из теоретико-графовых внутренних представлений программ. Однотипные компоненты могут быть объединены в функциональные блоки. Функциональные и инструментальные блоки реализуются как внешние библиотеки (dll либо элементы ActiveX) с учетом заданных интерфейсов взаимодействия с системой, а точнее с Внутренним Представлением Систе-

[*] Работа выполнена при финансовой поддержке Российского Фонда Фундаментальных Исследований (гранты № 02-07-90409 и 03-07-06-112).

мы Прогресс-2. Регистрация либо загрузка соответствующих компонентов происходит во время запуска системы, либо во время начала работы над заданным СЦЕНАРИЕМ.

Исходя из выше сказанного, система рассматривается как КОНСТРУКТОР для построения прототипа компилятора, кирпичиками которого являются различные функциональные и инструментальные компоненты. С помощью заданного СЦЕНАРИЯ пользователь сможет построить, исполнить либо исследовать прототип построенного компилятора.

Компоненты системы могут быть реализованы как отдельные процедуры внешней библиотеки, взаимодействующие с заданными объектами внутреннего представления системы. Либо компонент может быть определен как тип СООБЩЕНИЯ плюс множества ОБРАБОТЧИКОВ-ДЕЙСТВИЙ на данное сообщение для определенных типов объектов внутреннего представления. Для активизации данного типа компонента достаточно передать сообщение объекту внутреннего представления, обычно вершине какого-либо графового представления программы.

ЯДРО И ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ СИСТЕМЫ

Требования и общая модель

Основным компонентом системы является расширяемое внутреннее представление системы (ВП), которое служит для связи между компонентами. Главными целями при его проектировании являлись *универсальность (общность), расширяемость и гибкость*. Данное представление должно было бы позволить переводить во внутреннее отображение многие широко распространенные языки программирования, а также обобщить различные теоретико-графовые внутренние представления, разработанные за последнее время.

Модель разработанного внутреннего представления можно разделить на две составляющие: Языковая составляющая и Графовая составляющая.

Языковая составляющая ВП — это множество синтаксических и семантических объектов, иерархически упорядоченных, отображающих большинство общих конструкций входных языков. Все объекты распределяются на области видимости, тем самым ВП включает в себя таблицы объектов для каждой области видимости и операторную часть. Операторная часть представлена в виде абстрактного синтаксического графа, узлы которого делятся на семантические и управляющие вершины. К семантическим вер-

шинам относятся if-вершины, loop-вершины, или операторные вершины. К управляющим относятся вершины-директивы сценария, или дополнительные вершины ВП. Множество конструкций, соответствующих большинству конструкций императивных языков программирования, расширено операторами с параллельной семантикой исполнения.

К графовой составляющей ВП относятся различные теоретико-графовые представления программ, такие как управляющие граф, ГПЗ, граф зависимостей по данным и т.д. Пользователь может реализовать свое графовое представление и встроить в систему, при условии, что типы вершин графа будут унаследованы от базового типа Node_Object, который, в свою очередь, унаследован от основного класса OMA_Object.

Базовые классы и механизм передачи сообщений

Базовым классом для всех объектов ВП системы является класс OMA_Object. Основным атрибутом данного класса является указатель на объект типа OMA_ObjectType(*). Классы, наследованные от класса OMA_ObjectType, определяют типы различных объектов ВП и содержат локальные кучи объектов OMA_Object данного типа, что в частности позволяет динамически вести контроль выделения и удаления памяти. Внутренняя реализация класса (*) позволяет вести регистрацию всех наследуемых от него типов, динамического создания объектов и контроль типов.

При рассмотрении интерфейса класса OMA_Object:

```
class  PUBLIC OMA_Object: public OMA_Message{
public:
  static OMA_ObjectType *Type();
...
private:
  OMA_Act *act;
public:
  OMA_ObjectType *ObjectType() const
 …
//
  void Act(OMA_Message &msg)
OMA_Act *SetAct(OMA_Act *new_act=NULL)
…
static void ClassAct(OMA_Object *obj, OMA_Message &msg);};
```

помимо указателя на тип объекта можно выделить дополнительную составную часть атрибутов и методов, связанных с обработкой СООБЩЕНИЙ

(объекты, наследованные от класса OMA_Message). Данные СООБЩЕНИЯ могут быть реализацией какого-либо функционального компонента в виде прохода объектов ВП. При этом определенным типам объектов ВП устанавливается определенный обработчик выше указанного СООБЩЕНИЯ.

Рассмотрим использование механизма передачи сообщений на пример вычисление арифметического выражения. Допустим, что входное арифметическое выражение

```
(5+10-25)*2
```

было переведено во ВП в виде бинарного дерева. Вершины данного дерева являются экземплярами класса

```
class Exp_Node {        // должен быть наследован от OMA_Object или, если мы хотим воспользоваться средствами визуализации графовых представлений от Node_Object

// Предположим второе

…

Uint type; // 0 — если числовая константа 1- если сложение 2 — минус и т.д.

OMA_Variant val;

// значение числовой константы Описание класса OMA_Variant смотри ниже

…

}
```

Переменная

```
Exp_Node* Root //указатель на вершину
```

указывает на вершину.

Определяем новый тип сообщение

```
Class Calc_Exp: public OMA_Message{

…

  OMA_Variant curVal;

}
```

и определяем обработчик сообщений для типа вершин Exp_Node для сообщения Calc_Exp.

```
Static void OnCalcExp(Exp_Node* c, Calc_Exp & msg)
{ if (!c->type) msg.curVal=val;
    else
  {     c->son[0]->Act(msg);
            OMA_Varinat left=msg.curVal;
            c->son[1]->Act(msg);
        if (c->type==1) msg.curVal=left+msg.curVal;
        else
        … // Для остальных операций
  }
}
```

Теперь необходимо подключить обработчик сообщений к типу вершин Exp_Node

```
Exp_Node::Type()->SetAct(OnCalcExp, Calc_Exp::Type());
```

и запустить проход, передав сообщение вершине дерева

```
Calc_Exp msg;
Root->Act(msg);
```

После обработки сообщения в поле msg.curVal мы получим значение нашего арифметического выражения.

Введение механизма передачи СООБЩЕНИЙ позволяет легко переносить и модифицировать уже реализованные библиотечные компоненты.

Допустим, что кем-то реализован алгоритм. Описаны множество типов объектов ВП, с которыми реализация оперирует. Тогда, во-первых, данный алгоритм сможет обрабатывать введенные новые типы объектов ВП (при условии, что они наследованы от описанных в реализации алгоритма). А во-вторых, можно просто подключить свой обработчик сообщений для определенного типы вершины, не переписывая всю реализацию.

Дополнительные классы, типы данных, системные библиотеки

Основной целью при проектировании ВП ставилась его расширяемость. Поэтому ВП организовано в виде иерархии классов и библиотеки работы с ними. При разработке новых функциональных компонентов системы пользователь может либо по-своему интерпретировать зарезервированное поле, присутствующее в любом из классов, либо наследовать от уже существующих классов новые типы объектов.

Если говорить о языковой составляющей ВП, то базовым классом для всех внутренних объектов является класс UIFFEO_Object (унаследованный от базового класса OMA_Object), а для объектов операторной части — класс UIFFE_Object. Любой объект ВП содержит атрибуты, необходимые для визуализации программы. Для каждого объекта хранятся координаты его текстового прообраза — это номер строки и номер литеры в строке по текстовому файлу, для начальной и конечной литеры конструкции в тексте исходной программы. К атрибутам визуализации также можно отнести идентификатор исходной программы, вид объекта, цикла, безусловного перехода и информацию о переименование объектов в тексте исходной программы.

Помимо основных объектов в ВП введены системно зависимые объекты, так называемые директивы компилятора. Они могут быть установлены пользователем между запуском различных частей компилятора и указывать системе на вызов каких-либо библиотечных алгоритмов, функций и т.д. (в основном это объекты интерпретатора Сценария).

Помимо реализации типов объектов ВП в системной библиотеке реализованы различные структуры данных: массивы, неограниченные массивы, упорядоченные списки, древовидные структуры данных и т.д. Данные структуры и их реализации спроектированы с учетом поддержки механизма передачи сообщений, что может упростить реализацию новых функциональных компонентов.

Дополнительно, в системной библиотеке реализован класс OMA_Variant, инкапсулирующий в себя различные типы данных: integer, uint, float, double, char и т.д., а также указатель на базовый класс OMA_Object. Вариантный тип может быть использован, когда тип объекта явно не определен, а типизация происходит в период выполнения программы. Более детально это будет видно при описании Языка Сценариев.

Системная библиотека, которая автоматически подключается при запуске интерпретатора Сценариев, предоставляет также функции для работы с командной строкой, файловой системой, окнами и консолями, функции для

обработки ошибок, для работы с внешними библиотеками, хэш-таблицами и механизмы обработки текстовых файлов.

СЦЕНАРИЙ

Для создания своего собственного прототипа компилятора, пользователю необходимо определить СЦЕНАРИЙ его работы. Иными словами, пользователю необходимо задать набор программных библиотек, которыми он будет пользоваться, определить множество типов объектов внутреннего представления и определить множество и порядок исполнения функциональных компонентов (далее ПРОХОДОВ). Это может быть либо парсер, либо алгоритм оптимизации или преобразования дерева, либо компонент визуализации, например, Графа Программных Зависимостей. Ниже приведен возможный сценарий возможного прототипа компилятора “Конвертор с языка Паскаль в С”:

```
lib Pas=libPascal;libMy;Gr=libGraph;
pass Pas.Translate {
  UIFFE_Assign=libMy.OnTranslate_UIFFE_Assign;
};
pass Gr.BuildGraph;
pass Gr.VisCGraph;
$Module:=Pas.PASP_OpenFile(@1);
$Module:=PAS.PASP_TextParse($Module);
if ($Error ) {
        $OutputError(ErrMes);
}
else
{
          $Root:=Gr.Build_CGraph($Module);
          $Window_Title:="Control Graph";
          Gr.Out_Graph($Root);
          Run $Module(PAS.Translate); (***)
          IF ($Error) {
                  $OutputError($ErrorMes) ;
          }
}
end.
```

Далее разберем пример подробнее.

Декларация библиотек и сообщений

В данном примере можно увидеть основные объекты и операторы Языка Описания Сценариев (ЯОС). В начале сценария пользователю необходимо определить, какими программными библиотеками он будет пользоваться. Данные библиотеки он может взять из предложенных в системе либо написать свою, которую можно включить в сценарий. В нашем примере можно видеть, что используются три библиотеки: libPascal.dll (библиотека парсеров с языков Паскль и С), libMy.dll (дополнительная библиотека с измененным обработчиком сообщений) и libGraph.dll (библиотека работы с графовыми представлениями). Для задания используемых библиотек используется блок **lib**. Синонимы PAS и GR, заданные при объявлении библиотек, могут в дальнейшем использоваться пользователем для сокращения записи вызываемых функциональных и инструментальных библиотек. При подключении библиотеки интерпретатор автоматически вызывает процедуру *<Lib_Name>_Start* , в которой программист, создающий библиотеку, может инициализировать необходимые для работы библиотечных функций данные. Помимо этого, при завершении сценария интерпретатор вызывает для каждой загруженной библиотеки процедуру *<Lib_Name>_Finish*, где можно вставить вызовы деструкторов используемых объектов.

С помощью декларации **pass** пользователь объявляет используемые в дальнейшем проходы из библиотек. Проходы — это вид СООБЩЕНИЙ, описанный выше. Для проведения необходимых инициализирующих действий для проходов можно определить библиотечную функцию *<Pass_Name>_Init*, которая будет вызываться интерпретатором при объявлении прохода. Кроме этого, пользователь во время декларации библиотечного прохода может переопределить обработчик сообщений для конкретных типов объектов ВП системы. Так, в выше приведенном примере предлагается для сообщения Translate, определенного в библиотеке libPascal, имеющей синоним PAS, переопределить обработчик сообщений для объектов ВП типа UIFFE_Assign (присваивание) на другой, из пользовательской библиотеки libMy. Теперь, например, на выходе в С-коде вместо текста

```
A= b+c;
```

будем генерировать, к примеру, следующее:

```
A= b+ c; // Input data b & c; Output data A.
```

Тем самым, мы, не затрагивая всю реализацию С-кодогенератора и дерева разбора, поменяли лишь обработчик сообщений для вершин типа

UIFFE_Assign (присваивание). А все остальные обработчики остались библиотечными.

Здесь необходимо указать, что специальные библиотечные функции должны иметь определенную сигнатуру.

- Функция инициализации библиотеки — ***void <LibName>_Init ()***
- Функции деинициализации библиотек — ***void <LibName>_Finish ()***
- Функции инициализации СООБЩЕНИЙ — ***void <PASS_Name>_Init()***
- Обработчики СООБЩЕНИЙ —
  `void <Message_Handler>(OMA_Object *c, OMA_Message &msg)`

  Уже внутри реализации обработчика необходимо проверять конкретные типы входных параметров.
  Рассмотрим пример реализации переопределенного обработчика из нашего скрипта:

```
__declspec(dllexport)  void  OnTranslate_UIFFE_Assign(OMA_Object  *c,
OMA_Message &msg)
{
      if (msg.IsOf(Translate::Type()))
           OnTranslate((UIFFE_Assign*)c,(Translate&)msg);
      else
           c->Act(msg);
}
```

  Здесь OnTranslate — это статическая процедура с конкретными действиями обработчика. В начале происходит проверка типа сообщения msg. Если данный объект типа Translate — это сообщение-проход, отвечающее за кодогенерацию, то выполняем свои действия. Иначе объекту OMA_Object передаем сообщение msg. Конечно, здесь необходимо было бы дополнительно проверить тип объекта *c* на его принадлежность к объектам типа UIFFE_Assign, но в этом нет необходимости, если мы точно знаем, что данный обработчик определен только для объектов типа UIFFE_Assign.
- Функциональный и инструментальный компонент —

  `OMA_Variant <Func_Name>(OMA_Variant& in)`

  Как видно из сигнатуры, тип входного параметра и результата имеют вариантный тип. Данный подход позволяет создавать практически различные компоненты, не усложняя реализацию интерпретатора, но на-

кладывает определенные требования к документированию библиотек системы. Необходимо точно указать, какой тип передается на входе в вариантной переменной, и какой — на выходе. В нашем скрипте, например, компонент Gr.Build_CGraph (построение управляющего графа) на входе должен получить указатель на объект типа UIFFEO_Module (класс, инкапсулирующий в себе указатели на объекты транслируемого модуля и указатель на вершины дерева разбора). А на выходе получаем указатель на вершину управляющего графа типа, наследованного от класса Node_Object.

Другими объектами ЯОС являются переменные. Имена переменных начинаются со знака $. Все переменные имеют вариантный тип OMA_Variant. Нет необходимости явно объявлять переменную. Она создается в момент первого присваивания значения переменной. Переменные можно изменять не только в скрипте, но и внутри библиотечных компонентов с помощью библиотечных функций:

```
OMA_Variant Get_Engine_Variable(const char* name)
OMA_Variant Set_Engine_Variable(const char* name,
OMA_Variant& val)
```

Вторая функция возвращает старое значение переменной, если она существует, либо создает новую переменную с именем name и возвращает новое значение.

В нашем примере переменная $Window_Title используется компонентом Build_CGraph для указания имени GUI окна компонента. Если переменная не задана, то задается заголовок по умолчанию (надпись “Graph”).

Все переменные делятся на

- пользовательские, обычные переменные;
- системные переменные, существующие по умолчанию. Например, переменная $Error содержит код последней ошибки, а $ErrMes — текстовое представление;
- аргументы командной строки. Имеют вид @n, где n — номер аргумента начиная с единицы.

Операторная часть

Остальные конструкции ЯОС представляют операторную часть языка. Ниже описаны эти операторы.

Оператор вызова библиотечного компонента. В нашем примере это вызов процедур Pas.PASP_OpenFile, PASP_TextParse, Gr.Build_CGraph, Gr.Out_Graph. Первые два компонента отвечают за открытие текстового файла и разбор Паскаль-программы. Последние два используются для построения Управляющего графа программы и вывода его на экран. Все такие компоненты имеют одинаковую сигнатуру

```
OMA_Variant <Proc_Name>(OMA_Variant& in).
```

Видно, что для передачи и возврата данных используется вариантный тип, поэтому при описании создаваемого компонента пользователь должен будет явно указать: что скрывается за вариантными параметрами. Данное требование должно облегчить процесс создания новых функциональных компонентов. Например, известно, что компонент визуализации произвольного графа Gr.Out_Graph на входе должен получить указатель на вершину типа Node_Object. Соответственно, если мы хотим воспользоваться данным компонентом, то все вершины нашего графового представления должны наследоваться от типа Node_Object.

Функциональный компонент, как видно выше, может возвращать значение, которое может быть присвоено переменной сценария. Переменная сценария всегда имеет вариантный тип, и имя ее начинается со знака $. Переменные сценария служат для передачи данных между компонентами либо для вывода внутрисистемной информации.

С помощью **оператора IF_ELSE** можно определять поток управления в сценарии.

И последний оператор — это **оператор RUN**. Семантика данного оператора определяется как передача определенного сообщения объекту ВП. В нашем примере в строке (***) объекту $Module передается сообщение Translate, где переменная $Module содержит указатель на объект типа UIFFEO_Module, что можно рассматривать как корень языкового компонента ВП.

ЗАКЛЮЧЕНИЕ

Данная публикация дает краткое представление об основных компонентах системы и ее архитектурных особенностях. Более детальное описание с примерами можно найти на сайте разработчиков системы `http://pco.iis.nsk.su/progress`.

На текущем этапе работы идет создание программных библиотек для системы, их наполнение различными функциональными компонентами. Уже готовы такие компоненты системы, как парсеры с языка Паскаль и С, кодогенераторы в язык С, построение управляющего графа программы и его визуализации. Набор данных библиотек постоянно пополняется как авторами системы, так и некоторыми сторонними пользователями.

СПИСОК ЛИТЕРАТУРЫ

1. Маркин В.А., Маркина С.А. Проект системы для быстрого прототипирования распараллеливающего компилятора. Универсальное внутреннее представление системы // Современные проблемы конструирования программ. —Новосибирск, 2002.
2. Kasyanov V.N., Evstigneev V.A. The PROGRESS program manipulation system // Proc. of the Intern. Conf. “Parallel Computer Technologies”. — Obninsk, 1993. — Vol.3. — P. 651–656
3. Kasyanov V.N., Evstigneev V.A., Gorodniaia L. The system PROGRESS is a tool for parallelizing compiler prototyping // Proc. of 8-th SIAM Conf. on Parallel Processing for Scientific Computing (PPSC-97). — Minneapolis, 1997. — P. 301–306.

А. Л. Серебренников

СТАНДАРТНЫЕ И НОВЫЕ ПОДХОДЫ К АРХИТЕКТУРЕ И МЕТОДАМ ОБУЧЕНИЯ В СРЕДЕ SIGNIFICO, ОСНОВНЫЕ НАПРАВЛЕНИЯ РАЗВИТИЯ СРЕДЫ[*]

1. ВВЕДЕНИЕ

В данной работе описывается проектируемая интегрированная среда создания, обучения и диагностики нейросетей под названием Significo. Потенциальных пользователей среды Significo можно разделить на две категории. Первая категория пользователей — это исследователи как нейросетей, так и методов их обучения. Вторая категория — это разного рода аналитики.

В данное время на рынке программных продуктов широко представлены пакеты для использования нейросетевых технологий, но эти пакеты, как правило, предназначены для финансового прогнозирования. Доминирующее число существующих пакетов использует традиционные подходы и методы, имея только лишь продуманный и удобный интерфейс. Учитывая вышесказанное, становится явной востребованность в интегрированной среде, которая позволила бы не только специалистам в области нейросетевых технологий удобно проводить исследования, но и аналитикам анализировать различные предметные области.

Эта статья содержит шесть основных разделов.

- Задачи, решаемые с помощью нейросетевых технологий.
- Преобработка данных.
- Концепции передачи данных.
- Новые подходы к задаче проектирования структуры сети.
- Новые подходы к задаче обучения нейросетей.
- Интерфейсная часть.

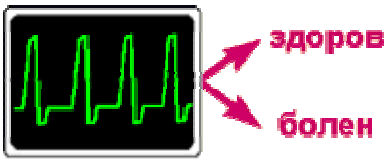
В этих разделах сначала будут рассматриваться стандартные подходы, а затем сравниваться с новыми по временным и качественным характеристикам.

[*] Работа выполнена при финансовой поддержке Министерства образования РФ (грант № E02-1.0-42).

2. ЗАДАЧИ, РЕШАЕМЫЕ С ПОМОЩЬЮ НЕЙРОСЕТЕВЫХ ТЕХНОЛОГИЙ

Классификация

Отметим, что задачи классификации (типа распознавания букв) очень плохо алгоритмизуются. Если в случае распознавания букв верный ответ очевиден для нас заранее, то в более сложных практических задачах обученная нейросеть выступает как эксперт, обладающий большим опытом и способный дать ответ на трудный вопрос.



Примером такой задачи служит медицинская диагностика, где сеть может учитывать большое количество числовых параметров (энцефалограмма, давление, вес и т.д.). Конечно, «мнение» сети в этом случае нельзя считать окончательным.

Классификация предприятий по степени их перспективности — это уже привычный способ использования нейросетей в практике западных компаний. При этом сеть также использует множество экономических показателей, сложным образом связанных между собой.

Нейросетевой подход особенно эффективен в задачах экспертной оценки по той причине, что он сочетает в себе способность компьютера к обработке чисел и способность мозга к обобщению и распознаванию. Говорят, что у хорошего врача способность к распознаванию в своей области столь велика, что он может провести приблизительную диагностику уже по внешнему виду пациента. Можно согласиться также, что опытный трейдер чувствует направление движения рынка по виду графика. Однако в первом случае все факторы наглядны, т. е. характеристики пациента мгновенно воспринимаются мозгом как «бледное лицо», «блеск в глазах» и т.д. Во втором же случае учитывается только один фактор, показанный на графике, — курс за определенный период времени. Нейросеть позволяет обрабатывать ог-

ромное количество факторов (до нескольких тысяч) независимо от их наглядности; это универсальный «хороший врач», который может поставить свой диагноз в любой области.

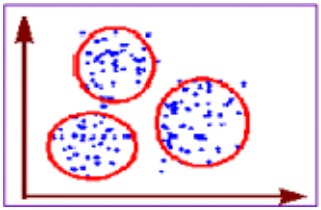
Кластеризация и поиск зависимостей

Помимо задач классификации, нейросети широко используются для поиска зависимостей в данных и кластеризации.

Например, нейросеть на основе методики МГУА (метод группового учета аргументов) позволяет на основе обучающей выборки построить зависимость одного параметра от других в виде полинома. Такая сеть может не только мгновенно выучить таблицу умножения, но и найти сложные скрытые зависимости в данных (например, финансовых), которые не обнаруживаются стандартными статистическими методами.

$$y = x_1^3 - 4x_3^2x_8^5 + x_3^2$$

Кластеризация — это разбиение набора примеров на несколько компактных областей (кластеров), причем число кластеров заранее неизвестно. Кластеризация позволяет представить неоднородные данные в более наглядном виде и использовать далее для исследования каждого кластера различные методы. Например, таким образом можно быстро выявить фальсифицированные страховые случаи или недобросовестные предприятия.



Прогнозирование

Задачи прогнозирования особенно важны для практики, в частности, для финансовых приложений, поэтому поясним способы применения нейросетей в этой области более подробно.

Рассмотрим практическую задачу, ответ в которой неочевиден, — задачу прогнозирования курса акций на 1 день вперед.

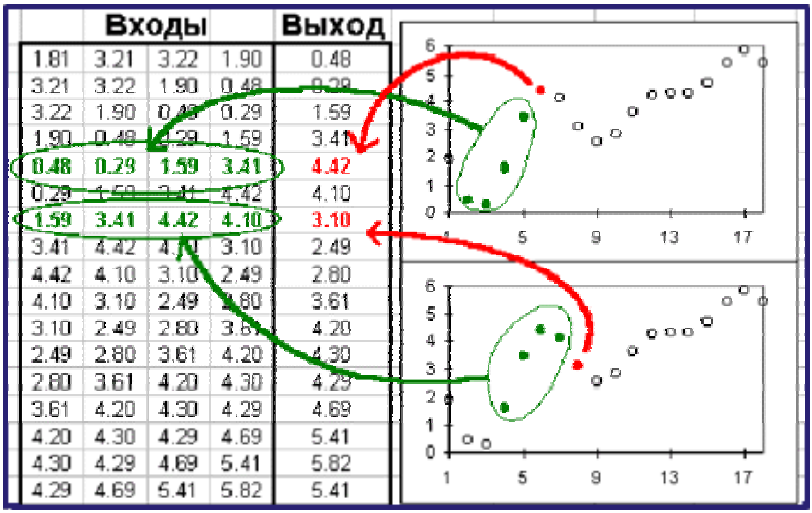
Пусть у нас имеется база данных, содержащая значения курса за последние 300 дней. Простейший вариант в данном случае — попытаться построить прогноз завтрашней цены на основе курсов за последние несколько дней. Понятно, что прогнозирующая сеть должна иметь всего один выход и столько входов, сколько предыдущих значений мы хотим использовать для прогноза, например, 4 последних значения. Составить обучающий пример

очень просто: входными значениями будут курсы за 4 последовательных дня, а желаемым выходом — известный нам курс в следующий день за этими четырьмя.

Если нейросеть совместима с какой-либо системой обработки электронных таблиц (например, Excel), то подготовка обучающей выборки состоит из следующих операций.

1. Скопировать столбец данных значений котировок в 4 соседних столбца.

2. Сдвинуть второй столбец на 1 ячейку вверх, третий столбец — на 2 ячейки вверх и т.д.



Смысл этой подготовки легко увидеть на рисунке: теперь каждая строка таблицы представляет собой обучающий пример, где первые 4 числа — входные значения сети, а пятое число — желаемое значение выхода. Исключение составляют последние 4 строки, где данных недостаточно, — эти строки не учитываются при тренировке. Заметим, что в четвертой снизу строке заданы все 4 входных значения, но неизвестно значение выхода. Именно к этой строке мы применим обученную сеть и получим прогноз на следующий день.

Как видно из этого примера, объем обучающей выборки зависит от выбранного нами количества входов. Если сделать 299 входов, то такая сеть

потенциально могла бы строить лучший прогноз, чем сеть с 4 входами, однако в этом случае мы имеем всего 1 обучающий пример, и обучение бессмысленно. При выборе числа входов следует учитывать это, выбирая разумный компромисс между глубиной предсказания (число входов) и качеством обучения (объем тренировочного набора).

3. ПРЕОБРАБОТКА ДАННЫХ

Преобработка данных имеет важнейшее значение, от возможностей этой части системы зависит в целом гибкость среды. Предполагается, что преобработка будет содержать 3 раздела.

Раздел символьной обработки конвертирует буквенно-численные выражения в числовой вид, который уже может быть обработан нейросетью. Например, понятия «холодно», «тепло», «жарко» можно представить числами 0, 1 и 2 соответственно.

Раздел логической обработки конвертирует данные числового вида, руководствуясь правилами пользователя, здесь имеются в виду правила типа «если—то—иначе».

Раздел спектральной обработки позволяет убрать из зависимостей помехи, например, различные циклически встречающиеся изменения в зависимостях.

4. НОВЫЕ ПОДХОДЫ К РЕШЕНИЮ ЗАДАЧ ПРОЕКТИРОВАНИЯ СТРУКТУРЫ НЕЙРОСЕТИ

Рассмотрим основные архитектуры нейросетей, имеющиеся на данный момент. В этой статье выделено несколько нейросетевых архитектур. Каждая архитектура наилучшим образом подходит для решения специфического круга задач.

Стандартные сети

Это стандартный тип сетей с обратным распространением, в котором каждый слой связан только с непосредственно предшествующим слоем. Это самая простая архитектура, и используется она в основном при решении задач распознавания образов.

Сети с обходными соединениями

В сетях с обратным распространением этого типа каждый слой связан со всеми предшествующими слоями. Задачи, выполняемые сетями с обходными путями, сходны с задачами, решаемыми стандартными сетями, но в некоторых случаях данные сети более помехоустойчивы.

Рекуррентные сети

Рекуррентные сети с обратным распространением ошибки очень хорошо подходят для решения задач прогнозирования. Например, часто применяются для финансовых биржевых предсказаний, поскольку эти сети могут запоминать последовательности. Благодаря этому свойству, такие сети являются прекрасным инструментом для работы с данными, представляющими собой временные серии.

Обычные сети с обратным распространением дают всегда в точности тот же самый ответ на один и тот же предъявляемый образ, тогда как ответ рекуррентной сети в подобных случаях будет зависеть от того, какие образы предъявлялись сети перед этим. Рекуррентные сети обладают долговременной памятью, построенной на внутренних нейронах.

Сети Ворда

Сети этого типа способны выделять различные свойства в данных, благодаря наличию в скрытом слое нескольких блоков, каждый из которых имеет свою передаточную функцию. Передаточные функции (обычно сигмоидного типа) служат для преобразования внутренней активности нейрона. Когда в разных блоках скрытого слоя используются разные передаточные функции, нейросеть оказывается способной выявлять новые свойства в предъявляемом образе.

Все вышеуказанные архитектуры являются архитектурами с обратным распространением ошибки и могут использовать разные способы подстройки весов: Простой, С Моментом и др. При Простом способе тренировки при предъявлении каждого примера происходит подстройка весов с заданной скоростью обучения. Подстройка С Моментом означает, что при модификации весов учитывается не только заданная скорость обучения, но и предыдущее изменение веса. TurboProp — это такой способ тренировки, при котором веса модифицируются после предъявления всех примеров (пакетная подстройка весов). Этот метод иногда работает гораздо быстрее, чем другие алгоритмы обратного распространения ошибки.

Все сети с обратным распространением ошибки используют Калибровку, что предотвращает возможность «переучивания» сети (и тем самым сильно сокращает время тренировки), а также улучшает способность сети обобщать информацию.

Сети Кохонена

Данные сети называют Саморганизующейся Картой Кохонена. Эта архитектура призвана решать задачи классификации. Сетям этого типа в процессе обучения не нужен «учитель», т.е. в процессе тренировки не требуется сообщать сети правильный ответ при предъявлении примера. Эти сети способны находить распределение данных на различные категории (классы).

Вероятностные нейронные сети

Вероятностные Нейронные Сети (ВНС) известны своей способностью обучаться на ограниченных наборах данных, причем для обучения нейросети достаточно однократного предъявления тренировочного набора! ВНС разделяет данные на указанное количество выходных категорий. Сеть ВНС зачастую способна работать уже после предъявления ей всего двух примеров из тренировочного набора, поэтому тренировка может осуществляться поэтапно.

Нейронные сети с общей регрессией

Подобно сетям ВНС, Нейронные Сети с Общей Регрессией (НСОР) известны своей способностью обучения в результате однократного предъявления тренировочных данных. Однако в отличие от сетей ВНС, которые классифицируют данные, сети НСОР способны предсказывать выходы с непрерывной амплитудой.

НСОР особенно полезны для аппроксимации непрерывных функций и могут строить многомерные поверхности, аппроксимирующие данные. Обнаружено, что для многих типов задач сети НСОР делают предсказания значительно лучше, чем сети с обратным распространением ошибки. Обращаем Ваше внимание: НСОР не использует приемов регрессионного анализа!

Для сетей ВНС и НСОР термин «Калибровка» означает оптимизацию параметра сглаживания, который используется в момент применения сетей. Модуль Калибровки с Генетическим Поиском использует генетический алгоритм для нахождения индивидуальных параметров сглаживания для каж-

дой входной переменной, что дает возможность получить сеть с очень хорошими обобщающими свойствами.

Полиномиальные сети

Полиномиальные сети представляют собой мощную архитектуру, называемую Методом группового учета аргумента (МГУА) или полиномиальными сетями. На самом деле, сеть МГУА непохожа на обычные сети с прямой связью, и изначально эта архитектура обычно не представлялась в виде сети. Сеть МГУА содержит в связях полиномиальные выражения и использует в некотором смысле аналогичный генетическим алгоритмам механизм принятия решения о том, сколько слоев необходимо построить. Результатом тренировки является возможность представить выход как полиномиальную функцию всех или части входов.

Новые подходы к проектированию архитектуры сети заключаются в том, что сеть должна быть разделена на отдельные взаимодействующие блоки с различным функциональным назначением. Как правило, аналитик, работающий над какой-либо предметной областью, может разделить поставленную перед ним задачу на функциональные блоки и определить между ними связь. При данной архитектуре функционального разделения общая сеть обладает двумя бесспорными плюсами. Первый плюс — это то, что одна большая сеть делится на несколько, это даёт выигрыш во времени обучения и потребляемых ресурсах.

В этом можно убедиться, рассмотрев график зависимости времени обучения нейросети на тестовом наборе обучающих пар от количества нейронов в нейросети (рис. 4.1).

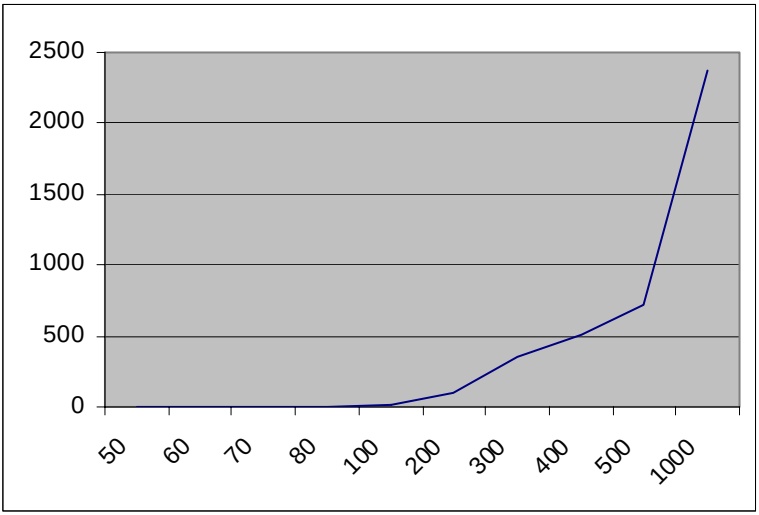


Рис. 4.1. График зависимости времени обучения нейросети от количества нейронов в ней

Второй плюс — это то, что отдельные блоки сети могут состоять из сетей различной архитектуры, что позволяет решать различные функциональные задачи сетями соответствующих архитектур.

5. НОВЫЕ ПОДХОДЫ К ОБУЧЕНИЮ НЕЙРОСЕТЕЙ

Обучение сети — это самый ресурсоёмкий процесс в работе с нейросетями. Оптимизация методов обучения позволяет открыть новые возможности нейросетей. Рассмотрим стандартные методы обучения нейросетей обратного распространения. Эти методы делятся в общем случае на 2 категории: рекурсивные и стохастические.

Долгое время не было теоретически обоснованного алгоритма для обучения многослойных искусственных нейронных сетей. А так как возможности представления с помощью однослойных нейронных сетей оказались весьма ограниченными, то и вся область в целом пришла в упадок.

Разработка алгоритма обратного распространения сыграла важную роль в возрождении интереса к искусственным нейронным сетям. Обратное распространение — это систематический метод для обучения многослойных

искусственных нейронных сетей. Он имеет солидное математическое обоснование. Несмотря на некоторые ограничения, процедура обратного распространения сильно расширила область проблем, в которых могут быть использованы искусственные нейронные сети, и убедительно продемонстрировала свою мощь.

Сетевые конфигурации

На рис. 5.1 показан нейрон, используемый в качестве основного строительного блока в сетях обратного распространения. Подается множество входов, идущих либо извне, либо от предшествующего слоя. Каждый из них умножается на вес, и произведения суммируются. Эта сумма, обозначаемая NET, должна быть вычислена для каждого нейрона сети. После того как величина NET вычислена, она модифицируется с помощью активационной функции и получается сигнал OUT.

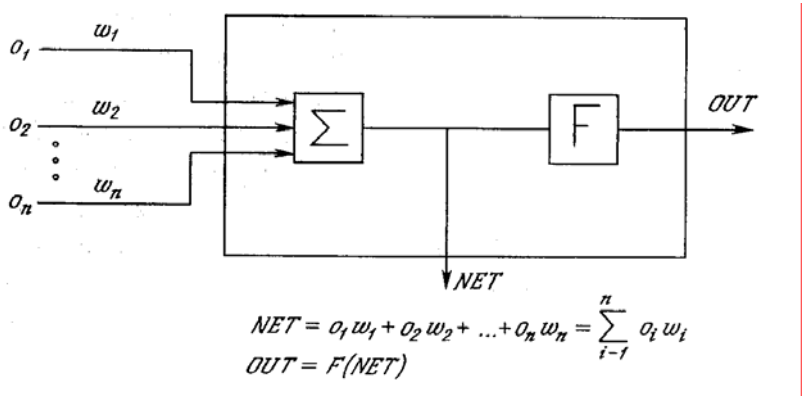


Рис. 5.1. Искусственный нейрон с активационнной функцией

На рис. 5.2 показана активационная функция, обычно используемая для обратного распространения.

$$\mathrm{OUT} = \frac{1}{1+\mathrm{e}^{-\mathrm{NET}}}. \qquad (5.1)$$

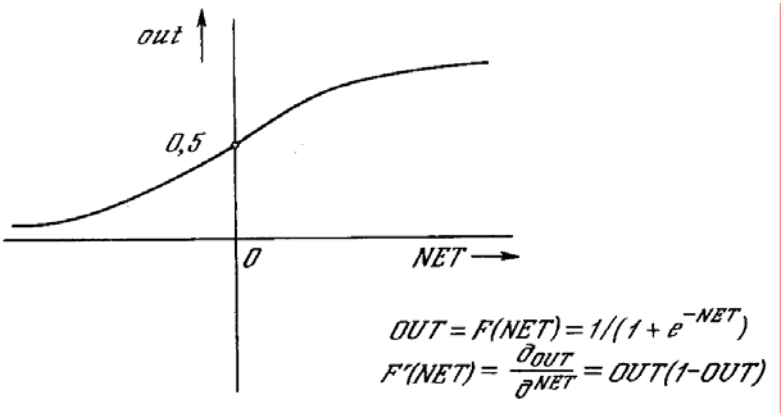


Рис. 5.2. Сигмоидальная активационная функция

Как показывает уравнение (5.2), эта функция, называемая *сигмоидом,* весьма удобна, так как имеет простую производную, что используется при реализации алгоритма обратного распространения.

$$\frac{\partial \mathrm{OUT}}{\partial \mathrm{NET}} = \mathrm{OUT}(1-\mathrm{OUT}). \qquad (5.2)$$

Сигмоид, который иногда называется также *логистической* или *сжимающей функцией,* сужает диапазон изменения NET так, что значение OUT лежит между нулем и единицей. Как указывалось выше, многослойные нейронные сети обладают большей представляющей мощностью, чем однослойные, только в случае присутствия нелинейности. Сжимающая функция обеспечивает требуемую нелинейность.

В действительности имеется множество функций, которые могли бы быть использованы. Для алгоритма обратного распространения требуется лишь, чтобы функция была всюду дифференцируема. Сигмоид удовлетворяет этому требованию. Его дополнительное преимущество состоит в автоматическом контроле усиления. Для слабых сигналов (величина NET близка к нулю) кривая вход-выход имеет сильный наклон, дающий большое усиление. Когда величина сигнала становится больше, усиление падает. Таким образом, большие сигналы воспринимаются сетью без насыщения, а слабые сигналы проходят по сети без чрезмерного ослабления.

Многослойная сеть

На рис. 5.3 изображена многослойная сеть, которая может обучаться с помощью процедуры обратного распространения (для ясности рисунок упрощен). Первый слой нейронов (соединенный с входами) служит лишь в качестве распределительных точек, суммирования входов здесь не производится. Входной сигнал просто проходит через них к весам на их выходах. А каждый нейрон последующих слоев выдает сигналы NET и OUT, как описано выше.

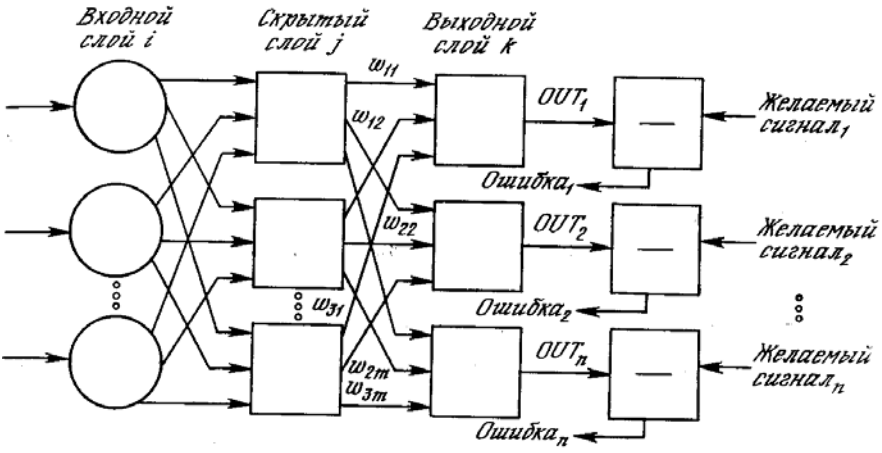


Рис. 5.3. Двухслойная сеть обратного распространения (ε — желаемый сигнал)

В литературе нет единообразия относительно того, как считать число слоев в таких сетях. Одни авторы используют число слоев нейронов (включая несуммирующий входной слой), другие — число слоев весов. Так как последнее определение функционально описательное, то оно будет использоваться на протяжении всей статьи. Согласно этому определению, сеть на рис. 5.3 рассматривается как двухслойная. Нейрон объединен с множеством весов, присоединенных к его входу. Таким образом, веса первого слоя оканчиваются на нейронах первого слоя. Вход распределительного слоя считается нулевым слоем.

Процедура обратного распространения применима к сетям с любым числом слоев. Однако для того чтобы продемонстрировать алгоритм, достаточно двух слоев. Сейчас будут рассматриваться лишь сети прямого действия, хотя обратное распространение применимо и к сетям с обратными связями.

Обучение сети обратного распространения требует выполнения следующих операций.

1. Выбрать очередную обучающую пару из обучающего множества; подать входной вектор на вход сети.
2. Вычислить выход сети.
3. Вычислить разность между выходом сети и требуемым выходом (целевым вектором обучающей пары).
4. Подкорректировать веса сети так, чтобы минимизировать ошибку.
5. Повторять шаги с 1 по 4 для каждого вектора обучающего множества до тех пор, пока ошибка на всем множестве не достигнет приемлемого уровня.

Операции, выполняемые шагами 1 и 2, сходны с теми, которые выполняются при функционировании уже обученной сети, т. е. подается входной вектор и вычисляется получающийся выход. Вычисления выполняются послойно. На рис. 5.3 сначала вычисляются выходы нейронов слоя *j*, затем они используются в качестве входов слоя *k*, вычисляются выходы нейронов слоя *k*, которые и образуют выходной вектор сети.

На шаге 3 каждый из выходов сети, которые на рис. 5.3 обозначены OUT, вычитается из соответствующей компоненты целевого вектора, чтобы получить ошибку. Эта ошибка используется на шаге 4 для коррекции весов сети, причем знак и величина изменений весов определяются алгоритмом обучения (см. ниже).

После достаточного числа повторений этих четырех шагов разность между действительными выходами и целевыми выходами должна уменьшиться до приемлемой величины, при этом говорят, что сеть «обучилась». Теперь сеть используется для распознавания, и веса не изменяются.

На шаги 1 и 2 можно смотреть как на «проход вперед», так как сигнал распространяется по сети от входа к выходу. Шаги 3, 4 составляют «обратный проход», здесь вычисляемый сигнал ошибки распространяется обратно по сети и используется для подстройки весов. Эти два прохода теперь будут детализированы и выражены в более математической форме.

Проход вперед. Шаги 1 и 2 могут быть выражены в векторной форме следующим образом: подается входной вектор X, и на выходе получается вектор Y. Векторная пара вход-цель X и T берется из обучающего множества. Вычисления проводятся над вектором X, чтобы получить выходной вектор Y.

Как мы видели, вычисления в многослойных сетях выполняются слой за слоем, начиная с ближайшего к входу слоя. Величина NET каждого нейрона первого слоя вычисляется как взвешенная сумма входов нейрона. Затем активационная функция F «сжимает» NET и дает величину OUT для каждого нейрона в этом слое. Когда множество выходов слоя получено, оно является входным множеством для следующего слоя. Процесс повторяется слой за слоем, пока не будет получено заключительное множество выходов сети.

Этот процесс может быть выражен в сжатой форме с помощью векторной нотации. Веса между нейронами могут рассматриваться как матрица W. Например, вес от нейрона 8 в слое 2 к нейрону 5 слоя 3 обозначается $w_{8,5}$. Тогда NET-вектор слоя N может быть выражен не как сумма произведений, а как произведение X и W. В векторном обозначении N = XW. Покомпонентным применением функции F к NET-вектору N получается выходной вектор O. Таким образом, для данного слоя вычислительный процесс описывается следующим выражением:

$$O = F(XW). \tag{5.3}$$

Выходной вектор одного слоя является входным вектором для следующего, поэтому вычисление выходов последнего слоя требует применения уравнения (5.3) к каждому слою от входа сети к ее выходу.

Обратный проход. *Подстройка весов выходного слоя.* Так как для каждого нейрона выходного слоя задано целевое значение, то подстройка весов легко осуществляется с использованием модифицированного дельта-правила. Внутренние слои называют «скрытыми слоями», для их выходов не имеется целевых значений для сравнения. Поэтому обучение усложняется.

На рис. 5.4 показан процесс обучения для одного веса от нейрона p в скрытом слое j к нейрону q в выходном слое k. Выход нейрона слоя k, вычитаясь из целевого значения (Target), дает сигнал ошибки. Он умножается на производную сжимающей функции [OUT(1 — OUT)], вычисленную для этого нейрона слоя k, что дает величину δ:

$$\delta = OUT(1 — OUT)(Target — OUT). \tag{5.4}$$

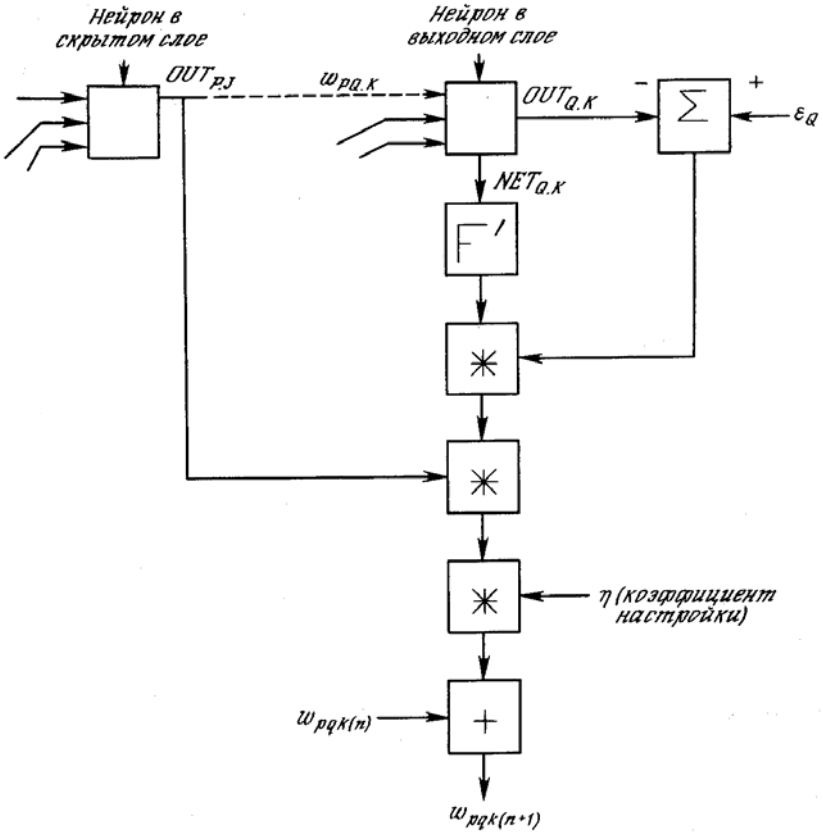


Рис. 5.4. Настройка веса в выходном слое

Затем δ умножается на величину OUT нейрона *j*, из которого выходит рассматриваемый вес. Это произведение, в свою очередь, умножается на коэффициент скорости обучения η (обычно от 0,01 до 1,0), и результат прибавляется к весу. Такая же процедура выполняется для каждого веса от нейрона скрытого слоя к нейрону в выходном слое.

Следующие уравнения иллюстрируют это вычисление:

$$\Delta w_{pq,k} = \eta\, \delta_{q,k}\, \mathrm{OUT} \tag{5.5}$$

$$w_{pq,k}(n+1) = w_{pq,k}(n) + \Delta w_{pq,k}, \tag{5.6}$$

где $w_{pq,k}(n)$ — величина веса от нейрона p в скрытом слое к нейрону q в выходном слое на шаге n (до коррекции); отметим, что индекс k относится к слою, в котором заканчивается данный вес, т. е. согласно принятому здесь соглашению, с которым он объединен; $w_{pq,k}(n+1)$ — величина веса на шаге n + 1 (после коррекции); $\delta_{q,k}$ — величина δ для нейрона q в выходном слое k; $OUT_{p,j}$ — величина OUT для нейрона p в скрытом слое j.

Подстройка весов скрытого слоя. Рассмотрим один нейрон в скрытом слое, предшествующем выходному слою. При проходе вперед этот нейрон передает свой выходной сигнал нейронам в выходном слое через соединяющие их веса. Во время обучения эти веса функционируют в обратном порядке, пропуская величину δ от выходного слоя назад к скрытому слою. Каждый из этих весов умножается на величину δ нейрона, к которому он присоединен в выходном слое. Величина δ, необходимая для нейрона скрытого слоя, получается суммированием всех таких произведений и умножением на производную сжимающей функции (см. рис. 5.5):

$$\delta_{q,k} = OUT_{p,j}(1 - OUT_{p,j})\left[\sum_{q} \delta_{q,k} w_{pq,k}\right] \qquad (5.7)$$

Когда значение δ получено, веса, питающие первый скрытый уровень, могут быть подкорректированы с помощью уравнений (5.5) и (5.6), где индексы модифицируются в соответствии со слоем.

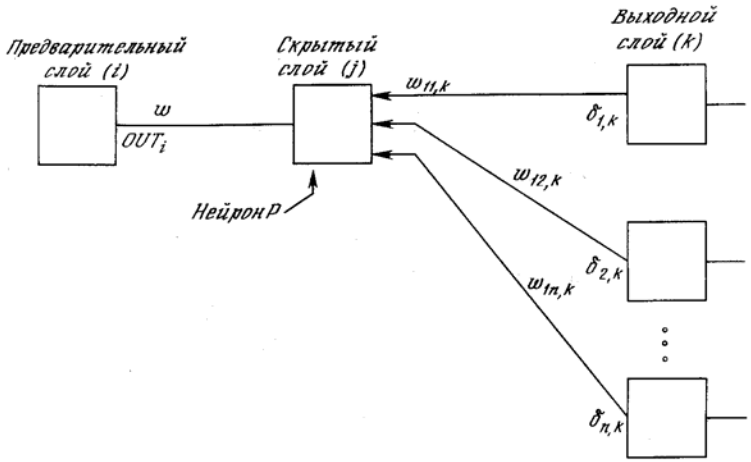


Рис. 5.5. Настройка веса в скрытом слое

Для каждого нейрона в данном скрытом слое должно быть вычислено δ и подстроены все веса, ассоциированные с этим слоем. Этот процесс повторяется слой за слоем по направлению к входу, пока все веса не будут подкорректированы

Стохастические методы

Стохастические методы полезны как для обучения искусственных нейронных сетей, так и для получения выхода от уже обученной сети. Стохастические методы обучения приносят большую пользу, позволяя исключать локальные минимумы в процессе обучения. Но с ними также связан ряд проблем.

Использование стохастических методов для получения выхода от уже обученной сети рассматривалось в работе [2].

Искусственная нейронная сеть обучается посредством некоторого процесса, модифицирующего ее веса. Если обучение успешно, то предъявление сети множества входных сигналов приводит к появлению желаемого множества выходных сигналов. Имеются два класса обучающих методов: детерминистский и стохастический.

Детерминистский метод обучения шаг за шагом осуществляет процедуру коррекции весов сети, основанную на использовании их текущих значений, а также величин входов, фактических выходов и желаемых выходов.

Стохастические методы обучения выполняют псевдослучайные изменения величин весов, сохраняя те изменения, которые ведут к улучшениям. Чтобы увидеть, как это может быть сделано, рассмотрим рис. 5.6, на котором изображена типичная сеть, в которой нейроны соединены с помощью весов. Выход нейрона является здесь взвешенной суммой его входов, которая преобразована с помощью нелинейной функции. Для обучения сети может быть использована следующая процедура.

1. Выбрать вес случайным образом и подкорректировать его на небольшое случайное значение. Предъявить множество входов, и вычислить получающиеся выходы.

2. Сравнить эти выходы с желаемыми выходами и вычислить величину разности между ними. Общепринятый метод состоит в нахождении разности между фактическим и желаемым выходами для каждого элемента обучаемой пары, возведение разностей в квадрат и нахождение суммы этих квадратов. Целью обучения является минимизация этой разности, часто называемой *целевой функцией.*

3. Выбрать вес случайным образом и подкорректировать его на небольшое случайное значение. Если коррекция помогает (уменьшает целевую функцию), то сохранить ее, в противном случае вернуться к первоначальному значению веса.

4. Повторять шаги с 1 до 3 до тех пор, пока сеть не будет обучена в достаточной степени.

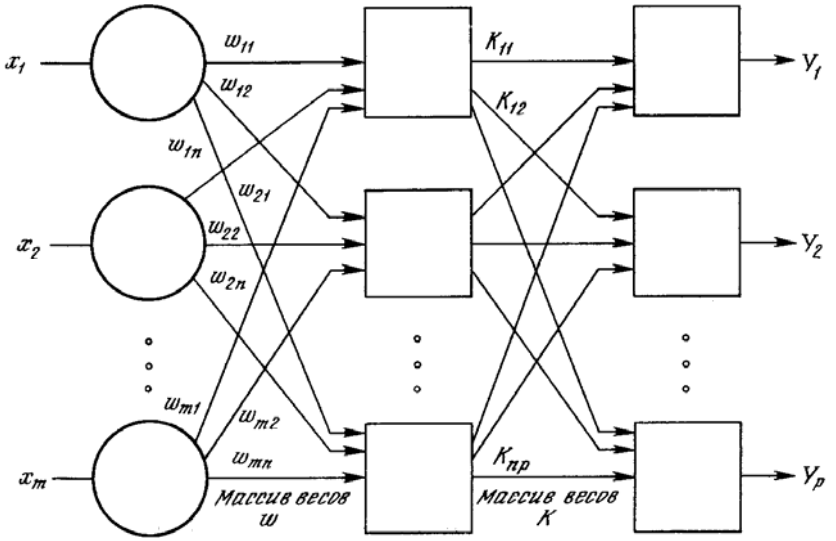


Рис. 5.6. Двухслойная сеть без обратных связей

Этот процесс стремится минимизировать целевую функцию, но может попасть, как в ловушку, в неудачное решение. На рис. 5.7 показано, как это может иметь место в системе с единственным весом. Допустим, что первоначально вес взят равным значению в точке А. Если случайные шаги по весу малы, то любые отклонения от точки А увеличивают целевую функцию и будут отвергнуты. Лучшее значение веса, принимаемое в точке В, никогда не будет найдено, и система будет поймана в ловушку локальным минимумом, вместо глобального минимума в точке В. Если же случайные коррекции веса очень велики, то как точка А, так и точка В будут часто посещаться, но то же самое справедливо и для каждой другой точки. Вес будет меняться так резко, что он никогда не установится в желаемом минимуме.

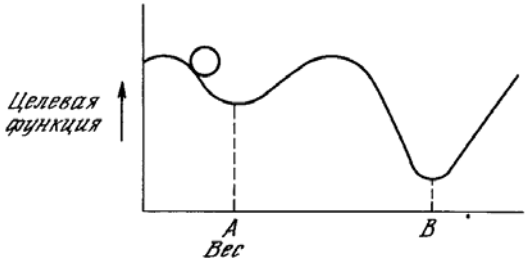


Рис.5.7. Проблема локальных минимумов

Полезная стратегия для избежания подобных проблем состоит в больших начальных шагах и постепенном уменьшении размера среднего случайного шага. Это позволяет сети вырываться из локальных минимумов и в то же время гарантирует окончательную стабилизацию сети.

Ловушки локальных минимумов досаждают всем алгоритмам обучения, основанным на поиске минимума, включая персептрон и сети обратного распространения, и представляют серьезную и широко распространенную трудность, которой часто не замечают. Стохастические методы позволяют решить эту проблему. Стратегия коррекции весов, вынуждающая веса принимать значение глобального оптимума в точке B, возможна.

В качестве объясняющей аналогии предположим, что на рис. 5.7 изображен шарик на поверхности в коробке. Если коробку сильно потрясти в горизонтальном направлении, то шарик будет быстро перекатываться от одного края к другому. Нигде не задерживаясь, в каждый момент шарик будет с равной вероятностью находиться в любой точке поверхности.

Если постепенно уменьшать силу встряхивания, то будет достигнуто условие, при котором шарик будет на короткое время «застревать» в точке B. При еще более слабом встряхивании шарик будет на короткое время останавливаться как в точке A, так и в точке B. При непрерывном уменьшении силы встряхивания будет достигнута критическая точка, когда сила встряхивания достаточна для перемещения шарика из точки A в точку B, но недостаточна для того, чтобы шарик мог передвинуться из B в A. Таким образом, окончательно шарик остановится в точке глобального минимума, когда амплитуда встряхивания уменьшится до нуля.

Искусственные нейронные сети могут обучаться по существу таким же образом посредством случайной коррекции весов. Вначале делаются большие случайные коррекции с сохранением только тех изменений весов, ко-

торые уменьшают целевую функцию. Затем средний размер шага постепенно уменьшается, и глобальный минимум в конце концов достигается.

Это сильно напоминает отжиг металла, поэтому для ее описания часто используют термин «имитация отжига». В металле, нагретом до температуры, превышающей его точку плавления, атомы находятся в сильном беспорядочном движении. Как и во всех физических системах, атомы стремятся к состоянию минимума энергии (единому кристаллу в данном случае), но при высоких температурах энергия атомных движений препятствует этому. В процессе постепенного охлаждения металла возникают все более низкоэнергетические состояния, пока в конце концов не будет достигнуто низшее из возможных состояний, глобальный минимум. В процессе отжига распределение энергетических уровней описывается следующим соотношением:

$$P(e) = \exp(-e/kT), \tag{5.8}$$

где $P(e)$ — вероятность того, что система находится в состоянии с энергией e; k — постоянная Больцмана; T — температура по шкале Кельвина.

При высоких температурах $P(e)$ приближается к единице для всех энергетических состояний. Таким образом, высокоэнергетическое состояние почти столь же вероятно, как и низкоэнергетическое. По мере уменьшения температуры вероятность высокоэнергетических состояний уменьшается по сравнению с низкоэнергетическими. При приближении температуры к нулю становится весьма маловероятным, чтобы система находилась в высокоэнергетическом состоянии.

Больцмановское обучение

Этот стохастический метод непосредственно применим к обучению искусственных нейронных сетей.

1. Определить переменную T, представляющую искусственную температуру. Придать T большое начальное значение.
2. Предъявить сети множество входов, и вычислить выходы и целевую функцию.
3. Дать случайное изменение весу, и пересчитать выход сети и изменение целевой функции в соответствии со сделанным изменением веса.
4. Если целевая функция уменьшилась (улучшилась), то сохранить изменение веса.

Если изменение веса приводит к увеличению целевой функции, то вероятность сохранения этого изменения вычисляется с помощью распределения Больцмана:

$$P(c) = \exp(-c/kT), \tag{5.9}$$

где $P(c)$ — вероятность изменения c в целевой функции; k — константа, аналогичная константе Больцмана, выбираемая в зависимости от задачи; T — искусственная температура.

Выбирается случайное число r из равномерного распределения от нуля до единицы. Если $P(c)$ больше, чем r, то изменение сохраняется, в противном случае величина веса возвращается к предыдущему значению.

Это позволяет системе делать случайный шаг в направлении, портящем целевую функцию, позволяя ей тем самым вырываться из локальных минимумов, где любой малый шаг увеличивает целевую функцию.

Для завершения больцмановского обучения повторяют шаги 3 и 4 для *каждого* из весов сети, постепенно уменьшая температуру T, пока не будет достигнуто допустимо низкое значение целевой функции. В этот момент предъявляется другой входной вектор и процесс обучения повторяется. Сеть обучается на всех векторах обучающего множества, с возможным повторением, пока целевая функция не станет допустимой для всех них.

Величина случайного изменения веса на шаге 3 может определяться различными способами. Например, подобно тепловой системе весовое изменение w может выбираться в соответствии с гауссовским распределением:

$$P(w) = \exp(-w^2/T^2), \tag{5.10}$$

где $P(w)$ — вероятность изменения веса на величину w, T — искусственная температура.

Такой выбор изменения веса приводит к системе, аналогичной [3].

Так как нужна величина изменения веса Δw, а не вероятность изменения веса, имеющего величину w, то метод Монте-Карло может быть использован следующим образом.

1. Найти кумулятивную вероятность, соответствующую $P(w)$. Это есть интеграл от $P(w)$ *в* пределах от 0 до w. Так как в данном случае $P(w)$ не может быть проинтегрирована аналитически, она должна интегрироваться численно, а результат необходимо затабулировать.

2. Выбрать случайное число из равномерного распределения на интервале (0,1). Используя эту величину в качестве значения $P(w)$, найти в таблице соответствующее значение для величины изменения веса.

Свойства машины Больцмана широко изучались. В работе [1] показано, что скорость уменьшения температуры должна быть обратно пропорциональна логарифму времени, чтобы была достигнута сходимость к глобальному минимуму. Скорость охлаждения в такой системе выражается следующим образом:

$$T(t) = \frac{T_0}{\log(1+t)}, \tag{5.11}$$

где $T(t)$ — искусственная температура как функция времени; T_0 — начальная искусственная температура; t — искусственное время.

Этот разочаровывающий результат предсказывает очень медленную скорость охлаждения (и данные вычисления). Этот вывод подтвердился экспериментально. Машины Больцмана часто требуют для обучения очень большого ресурса времени.

Обучение Коши

В работе [6] развит метод быстрого обучения подобных систем. В этом методе при вычислении величины шага распределение Больцмана заменяется на распределение Коши. Распределение Коши имеет, как показано на рис. 5.8, более длинные «хвосты», тем самым увеличивается вероятность больших шагов. В действительности распределение Коши имеет бесконечную (неопределенную) дисперсию. С помощью такого простого изменения максимальная скорость уменьшения температуры становится обратно пропорциональной линейной величине, а не логарифму, как для алгоритма обучения Больцмана. Это резко уменьшает время обучения. Эта связь может быть выражена следующим образом.

$$T(t) = \frac{T_0}{1+t}. \tag{5.12}$$

Распределение Коши имеет вид

$$P(x) = \frac{T(t)}{T(t)^2 + x^2}, \tag{5.13}$$

где $P(x)$ есть вероятность шага величины x.

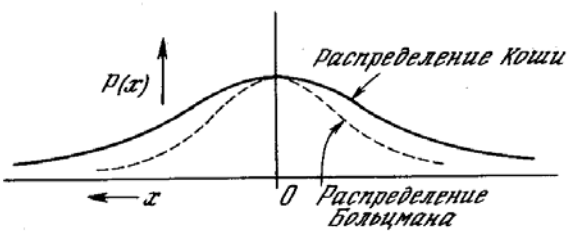


Рис. 5.8. Распределение Коши и распределение Больцмана

В уравнении (5.13) $P(x)$ может быть проинтегрирована стандартными методами. Решая относительно x, получаем:

$$x_c = \rho\, T(t)\, \mathrm{tg}(P(x)), \qquad (5.14)$$

где ρ — коэффициент скорости обучения; x_c — изменение веса.

Теперь применение метода Монте Карло становится очень простым. Для нахождения x в этом случае выбирается случайное число из равномерного распределения на открытом интервале $(-\pi/2, \pi/2)$ (необходимо ограничить функцию тангенса). Оно подставляется в формулу (5.13) в качестве $P(x)$, и с помощью текущей температуры вычисляется величина шага.

Метод искусственной теплоемкости

Несмотря на улучшение, достигаемое с помощью метода Коши, время обучения может оказаться все еще слишком большим. Способ, полученный на основе законов термодинамики, может быть использован для ускорения этого процесса. В этом методе скорость уменьшения температуры изменяется в соответствии с искусственной «теплоемкостью», вычисляемой в процессе обучения.

Во время отжига металла происходят фазовые переходы, связанные с дискретными изменениями уровней энергии. При каждом фазовом переходе может иметь место резкое изменение величины, называемой теплоемкостью. *Теплоемкость* определяется как скорость изменения температуры с энергией. Изменения теплоемкости происходят из-за попадания системы в локальные энергетические минимумы.

Искусственные нейронные сети проходят аналогичные фазы в процессе обучения. На границе фазового перехода искусственная теплоемкость может скачкообразно измениться. Эта псевдотеплоемкость определяется как

средняя скорость изменения температуры с целевой функцией. В примере шарика в коробке сильная начальная встряска делает среднюю величину целевой функции фактически не зависящей от малых изменений температуры, т. е. теплоемкость близка к константе. Аналогично при очень низких температурах система замерзает в точке минимума, так что теплоемкость снова близка к константе. Ясно, что в каждой из этих областей допустимы сильные изменения температуры, так как не происходит улучшения целевой функции.

При критических температурах небольшое уменьшение температуры приводит к большому изменению средней величины целевой функции. Возвращаясь к аналогии с шариком, при «температуре», когда шарик обладает достаточной средней энергией, чтобы перейти из А в В, но недостаточной — для перехода из В в А, средняя величина целевой функции испытывает скачкообразное изменение. В этих критических точках алгоритм должен изменять температуру очень медленно, чтобы гарантировать, что система не замерзнет случайно в точке А, оказавшись пойманной в локальный минимум. Критическая температура может быть обнаружена по резкому уменьшению искусственной теплоемкости, т. е. средней скорости изменения температуры с целевой функцией. При достижении критической температуры скорость изменения температуры должна замедляться, чтобы гарантировать сходимость к глобальному минимуму. При всех остальных температурах может без риска использоваться более высокая скорость снижения температуры, что приводит к значительному снижению времени обучения.

В процессе работы с рекурсивным методом было сделано добавление к нему в виде модуля управления скоростью обучения и модуля калибровки. При обучении сети при каждой итерации вычисляется значение калибровочной ошибки на всём наборе векторов обучения, далее в зависимости от градиента ошибки вычисляется значение скорости. При использовании данного оптимизационного метода общая скорость обучения выросла, и в то же время уменьшилась вероятность попадания процесса обучения в локальный минимум. Это можно заключить из проведённых опытов, состоящих в следующем: берутся две одинаковые нейросети и два одинаковых набора обучающих пар, производится обучение с одинаковым количеством рекурсий. В итоге опыта на стандартной процедуре обучения уровень ошибки получился равным 0,41, при обучении усовершенствованным методом при тех же условиях ошибка составила 0,992. Также (что не маловажно) у пользователя появилась возможность контролировать степень облученности сети. В дальнейшем предполагается автоматически определять количество итераций, необходимое для обучения сети с определённой точностью.

6. ИНТЕРФЕЙСНАЯ ЧАСТЬ

Так как предполагается использование среды как исследователями нейросетей так и различного рода аналитиками, интерфейс должен включать в себя как простоту и наглядность происходящих процессов, так и многофункциональность.

Наглядность происходящих процессов предполагает развитую систему визуализации с применением трехмерного моделирования. Также необходимо снабдить интерфейс удобным инструментом отслеживания процесса обучения с отображением истории различного рода ошибок в процессе обучения и других параметров.

Гибкость интерфейса предполагает, помимо возможности его изменения, возможность легкого манипулирования результатами различных операций с нейросетями. Предполагается использование OLE технологии для передачи результатов в пакет MS Office, что позволит пользователю доработать результаты (например, составить диаграмму или график) или передать данные для дальнейшей обработки в другие программы (например, MathCAD).

Среда Significo должна поддерживать несколько стандартов результатных файлов. Это файлы, содержащие результаты работы отдельных модулей среды. Планируется наличие проектных файлов, в которых будут связываться все поддерживаемые файлы среды.

7. ЗАКЛЮЧЕНИЕ

Практическим результатом данной работы является описание проектируемой интегрированной среды Significo и описание применяемых в ней технологий, видов сетей, алгоритмов обучения. Приведены новые подходы в области проектирования архитектуры сетей с практическим обоснованием выигрыша по времени обучения. Подробно представлены стандартные методы обучения нейросетей и наряду с ними новые подходы к обучению сетей и усовершенствованию стандартных методов обучения. Представлен результат проведённых опытов, которые показали, что средняя скорость обучения нейросети увеличилась примерно в 4 раза.

СПИСОК ЛИТЕРАТУРЫ

1. Geman S., Geman D. Stohastic relaxation, Gibbs distribution and Baysian restoration of images // IEEE Transactions on Pattern Analysis and Machine Intelligence. — 1984. — Vol. 6. — P. 721–741.
2. Hinton G. E., Sejnowski T. J. Learning and relearning in Boltzmann machines // Parallel distributed processing. — Cambridge, MA: MIT Press. — 1986. — Vol. 1. — P. 282–317..
3. Metropolis N., Rosenbluth A. W-.Rosenbluth M. N., Teller A. N., Teller E. Equations of state calculations by fast computing machines // J. of Chemistry and Physics. — 1953. — Vol. 21. — P. 1087–1091.
4. Материалы сайта `www.neuroproject.ru`
5. Уоссермен Ф. Нейрокомпьютерная техника: Теория и практика / Пер. на русский язык Ю. А. Зуев, В. А. Точенов. — М.: Мир. 1992. — 66 с.
6. Szu H., Hartley R. Fast Simulated annealing // Physics Letters. — 1987. — Vol. 1222, N 3,4. — P. 157–162.

Е. С. Черемушкин, Т. Г. Коновалова,
Ф. А. Мурзин, А. Э. Кель

СИСТЕМА РАСПОЗНАВАНИЯ ЦИС-ЭЛЕМЕНТОВ НА ПОСЛЕДОВАТЕЛЬНОСТЯХ ДНК*

ВВЕДЕНИЕ

В настоящее время проведены основные экспериментальные работы по секвенированию нуклеотидных последовательностей для достаточно большого числа организмов [1,2]. Для хранения получаемой первичной информации создан и постоянно пополняется ряд баз данных как специализированных, так и широкого профиля. В то же время, несмотря на наличие большого количества отсеквенированных последовательностей, наши представления о принципах их организации весьма ограничены. Поэтому одним из ведущих направлений молекулярной биологии в последнее время становится компьютерный анализ генетических текстов.

Проблематика понимания структурно-функциональной организации генома эукариот включает в себя широкий круг проблем. Наряду с такими вопросами, как распознавание интронов, экзонов или сайтов сплайсинга, существует все увеличивающийся круг задач, связанных с регуляцией транскрипции генов позвоночных. В последнее время появилось большое количество разнообразных данных (таких как SNP или паттерны экспрессии), позволяющих углубить понимание механизмов регуляции экспрессии генов. Одним из базовых понятий, занимающих ключевую роль в процессах транскрипции, является понятие транскрипционных факторов, которые представляют собой регуляторные белки, обладающие способностью распознавания специфических коротких участков ДНК. Поэтому детальному изучению и распознаванию соответствующих нуклеотидных фрагментов, называемых цис-элементами или сайтами связывания транскрипционных факторов (ССТФ или сайтами), отводится большое внимание. Несмотря на разнообразие подходов, проблема построения точных методов распознавания ССТФ в настоящее время не может считаться окончательно решенной. Причина этого состоит в большом разнообразии контекстных, физико-

* Работа выполнена при финансовой поддержке Министерства образования РФ (грант № E02-1.0-42).

химических и конформационных особенностей ССТФ; механизмов ДНК-белковых взаимодействий между ССТФ и транскрипционными факторами; специфичности контекста, окружающего ССТФ, степени консервативности нуклеотидного контекста в эволюции. Поэтому перспективным представляется применение методов, ориентированных в каждом конкретном случае на специфическую информацию, которой обладает биолог.

С этой целью был разработан комплекс алгоритмов идентификации цис-элементов и объектно-ориентированная среда, реализующая эти методы.

1. СУЩЕСТВУЮЩИЕ ДАННЫЕ

В распоряжении экспериментатора находится различная генетическая информация. От количества информации зависит качество распознавания.

Регуляторная последовательность. Это классический объект, которым обладает каждый экспериментатор. Задача, которую он решает с помощью определенного метода, — предсказание потенциальных сайтов определенного типа на этой последовательности. Отметим, что на основе только этой информации предсказание будет крайне неточным.

Гомологичные регуляторные последовательности разных организмов. В настоящее время экспериментатор зачастую обладает информацией об эволюционном сходстве изучаемого участка последовательности ДНК одного организма с некоторыми участками ДНК других организмов. Имея соответствующий метод, он может получить более точное предсказание цис-элементов на этом наборе последовательностей. В зависимости от уровня гомологии целесообразно использовать различные методы.

Функционально-связанные последовательности. Ввиду больших объемов аннотированной ДНК экспериментатор часто имеет набор функционально-зависимых промоторов генов, например, генов, вовлеченных в один биологический процесс. Таким образом, используя специфический метод, он имеет возможность предсказать транскрипционные факторы, регулирующие гены его выборки. Также может присутствовать несколько выборок.

Паттерн экспрессии генов. В настоящее время получили распространение паттерны экспрессии генов (expression pattern, microarray experiments). Используя специфическую технологию, экспериментатор получает набор чисел, соответствующих уровню экспрессии для большого

количества генов (порядка 20 000). Точность метода в данный момент не очень велика, но достаточна для широкого и эффективного использования этой информации. Зачастую одновременно используют результаты двух экспериментов: здоровой и больной клетки. Имея соответствующий метод, экспериментатор может предсказать цис-элементы, нарушающие нормальную деятельность организма, чтобы впоследствии воздействовать на соответствующие транскрипционные факторы.

Однонуклеотидные полиморфизмы. Однонуклеотидные полиморфизмы (Single-Nucleotide Polymorphism) — это различия в ДНК между индивидами одного вида. Они характеризуют индивидуальные особенности или особенности популяции (этноса). Полиморфизмы в регуляторных областях могут влиять на регуляцию. Таким образом, используя соответствующий метод, можно предсказывать изменение регуляции в связи с SNP.

2. МЕТОДЫ РАСПОЗНАВАНИЯ ЦИС-ЭЛЕМЕНТОВ

Авторами настоящей работы была создана объектно-ориентированная система, реализующая как созданные ранее, так и разработанные авторами методы. Данная среда является инструментарием для решения широкого круга задач распознавания цис-элементов.

2.1. Метод весовых матриц

Основная идея метода весовых матриц [3] заключается в приписывании четырех весов каждой позиции сайта в соответствии с четырьмя нуклеотидами A, T, G и C. Эти веса связаны с вероятностью появления конкретного нуклеотида в конкретной позиции.

Пусть $F = \left| f_{ij} \right|$ — 4×1 матрица нуклеотидных частот, f_{ij} — абсолютная частота встречаемости *i*-го нуклеотида на *j*-ой позиции в обучающей выборке выровненных нуклеотидных фрагментов, кодирующих известные сайты связывания (i=1,...,4; j=1,...,l). Элементы w_{ij} весовой матрицы W определяются соотношением:

$$w_{ij} = \ln\left(\frac{f_{ij}}{e_{ij}} + \frac{s}{100} \right) + c_i ,$$

где e_{ij} — ожидаемые частоты, соответствующие величинам f_{ij}, c_i — нуклеотидно-специфические константы, s — параметр сглаживания, измеряемый в процентах.

Таблица 1

Матрица нуклеотидных частот (F) и весовая матрица (W), вычисленные для кэп-сайта[1]

		позиции сайта							
		-2	-1	0	1	2	3	4	5
F:	‘A’	49	0	288	26	77	67	45	50
	‘C’	48	303	0	81	95	118	85	96
	‘G’	69	0	0	116	0	46	73	56
	‘T’	137	0	15	80	131	72	100	101
W:	‘A’	-1.1	-5.3	0.0	-1.5	-0.7	-0.6	-0.9	-0.8
	‘C’	-1.2	0.0	-5.2	-0.4	-0.5	0.0	-0.3	-0.2
	‘G’	-0.8	-5.3	-5.2	0.0	-4.6	-0.9	-0.4	-0.7
	‘T’	0.0	-5.3	-2.7	-0.3	0.0	-0.4	0.0	0.0

Процедура распознавания функционального сайта (характеризуемого весовой матрицей W) в произвольном нуклеотидном фрагменте длины L заключается в сопоставлении величины match и априорно заданного порогового значения $match^{(crit)}$:

$$match = 100 \times \frac{x - x_{\min}}{x_{\max} - x_{\min}},$$

где $x_{\max} = \sum_{j=1}^{L} \max_i(w_{ij})$, $x_{\min} = \sum_{j=1}^{L} \min_i(w_{ij})$, а значение x оценивает степень близости тестируемого фрагмента и обучающей выборки:

$$x = \sum_{j=1}^{L} w_{ij}.$$

[1] Идея метода весовых матриц заключается в приписывании четырех весов каждой позиции сайта в соответствии с четырьмя нуклеотидами A, T, G и C. Эти веса связаны с вероятностью появления конкретного нуклеотида в конкретной позиции.

Все потенциальные сайты в заданной нуклеотидной последовательности распознаются с помощью применения вышеизложенного алгоритма к каждому скользящему окну из этой последовательности.

2.2. Метод распознавания двойных сайтов

Сайты связывания некоторых транскрипционных факторов состоят из двух полусайтов с варьирующимся расстоянием между ними. Расстояние между полусайтами зависит от типа фактора, узнающего этот сайт. Полусайты могут иметь схожую структуру. Так как сайт состоит из 2-х консервативных доменов с варьирующим расстоянием между ними (рис. 1), то зададим double-core модели распознавания M_k следующим образом:

$$M_k = \langle m_1, m_2, d_1, d_2 \rangle$$

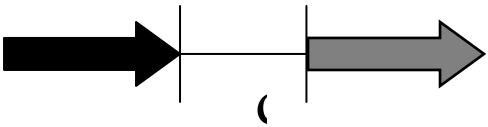


Рис.1. Сайт состоит из 2 консервативных доменов с варьирующимся расстоянием между ними

При этом m_1 и m_2 — весовые матрицы [3], d_1, d_2 — минимальное и максимальное расстояния между половинками сайтов. Пусть $w_1(i)$ и $w_2(j)$ веса m_1 и m_2 в позиции i и j соответственно на последовательности. Сайт считается распознанным, если вес $w = \dfrac{w_1(i) + w_2(j)}{2}$ больше заданного порога c и расстояние между половинками сайтов $d \in [d_1, d_2]$.

Распознавание сайтов NR будем производить следующим образом. Если на последовательности был распознан характерный полусайт, рассмотрим, какой максимальный вес w_k распознавания дает каждая из моделей M_k. Если модель M_k не распознана в данном районе, то считаем $w_k = 0$. Рассмотрим метод получения моделей M_k. Пусть $S = (S_1, ..., S_m)$ — обучающая выборка последовательностей сайтов. Для каждого подмножества

$S^{'} = (S_1^{'},...,S_m^{'})$ множества S зададим два набора подпоследовательностей $S^1 = (s_1^1,...,s_n^1)$ и $S^2 = (s_1^2,...,s_n^2)$, $s_i^1, s_i^2 \in S_i^{'}$, длина s_i^j равна 6.

Найдем с помощью широко используемой в биоинформатике процедуры гиббссамплинга [4] S^1 и S^2 такие, что s_i^1 похожи между собой в терминах расстояния между последовательностями, и s_i^2 похожи между собой. На основе S^1 и S^2 создадим соответствующие матрицы m_1 и m_2. Затем выберем расстояния $d_1 = \min_i \left(d(s_i^1, s_i^2) \right)$ и $d_2 = \max_i \left(d(s_i^1, s_i^2) \right)$. Выберем начальное подмножество $S_{[0]}$, называемое коровой выборкой.

Теперь построим модель $M_{[0]}$ и добавим в $S_{[0]}$ последовательность из $S \setminus S_{[0]}$, для которой вес $w_{[0]}$ модели $M_{[0]}$ максимален. Таким образом, получим модель $M_{[1]}$. Будем продолжать процедуру добавления до тех пор, пока вес $w_{[k]}$ превышает изначально заданный порог C.

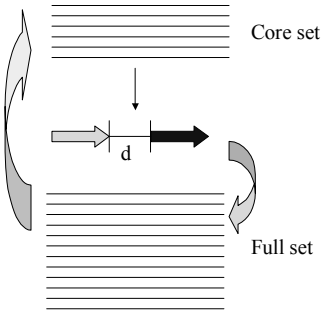


Рис.2. Процесс построения модели $M_{[i]}$

После окончания процедуры получим модель *M*, описывающую выборку *S*. Таким образом, получим различные модели $M_1,...,M_T$ для различных классов сайтов.

2.3. Филогенетический футпринт

Мы разработали алгоритм для выравнивания двух или более нуклеотидных последовательностей. Метод основывается на предположении, что в процессе эволюции цис-элементы более консервативны, чем другие участки промоторных последовательностей. Алгоритм схож с общепринятым алгоритмом Недельмана—Вунша[5]. Основные изменения сделаны в способе подсчета весов на нуклеотидные замены и в штрафах на делеции.

Штраф на делеции, при вставке гэпа в $\overline{S}^1$ между $k-1$ и k над позицией l в $\overline{S}^2$:

$$GAP\left(\overline{S}^1,\overline{S}^2,k,l\right)=\frac{G\left(\overline{S}^1,k\right)+R\left(\overline{S}^2,l\right)}{2}.$$

Штраф на замену:

$$SUB\left(\overline{S}^1,\overline{S}^2,k,l\right)=\mathrm{Z}\left(s_k^1,s_l^2\right),$$

где

$$G\left(\overline{S}^1,k\right)=\mathrm{Y}\left(s_{k-1}^1,s_k^1\right),$$

$$R\left(\overline{S}^2,l\right)=\frac{\mathrm{Y}\left(s_{l-1}^2,s_l^2\right)+\mathrm{Y}\left(s_l^2,s_{l+1}^2\right)}{2}$$

$$\mathrm{Y}\left(a,b\right)=\frac{C_{gap}}{N}+W_{gap}\cdot s_{gap}\left(a,b\right),$$

$$\mathrm{Z}\left(a,b\right)=\frac{\Delta}{N}\cdot C_{sub}-W_{sub}\cdot\frac{\sum_{i=1}^{3}\lambda_i\cdot s_i\left(a,b\right)}{\sum_{i=1}^{3}\lambda_i},\ \text{для}\ \ a,b\in\Sigma\times\Phi,$$

где

$$\Delta=\begin{cases}1,\gamma\left(a\right)\neq\gamma\left(b\right)\\0,\gamma\left(a\right)=\gamma\left(b\right)\end{cases},$$

$$s_{gap}\left(a,b\right)=\begin{cases}\left(\vec{\varphi}\left(a\right)+\vec{\varphi}\left(b\right)\right)^2,\gamma\left(a\right)=\gamma\left(b\right)\\\vec{\varphi}\left(a\right)^2+\vec{\varphi}\left(b\right)^2,\gamma\left(a\right)\neq\gamma\left(b\right)\end{cases},$$

$$s_1\left(a,b\right)=s_{gap}\left(a,b\right),$$

$$s_2(a,b) = \begin{cases} 0, m > C_{\min} \\ (C_{\min} - m)/C_{\min}, m \le C_{\min} \end{cases},$$

где

$$m = \min_i \left| \varphi_i(a) - \varphi_i(b) \right|,$$

$$s_3(a,b) = \max_i \left(\varphi_i(a) \cdot \varphi_i(b) \right),$$

$\gamma(a) \in \Sigma$ — нуклеотид,

$\vec{\varphi}(a) \in \Phi$ — вектор весов для матрицы,

C_{corr}, C_{gap}, W_{corr}, W_{gap}, λ_i — константы,

N — количество последовательностей.

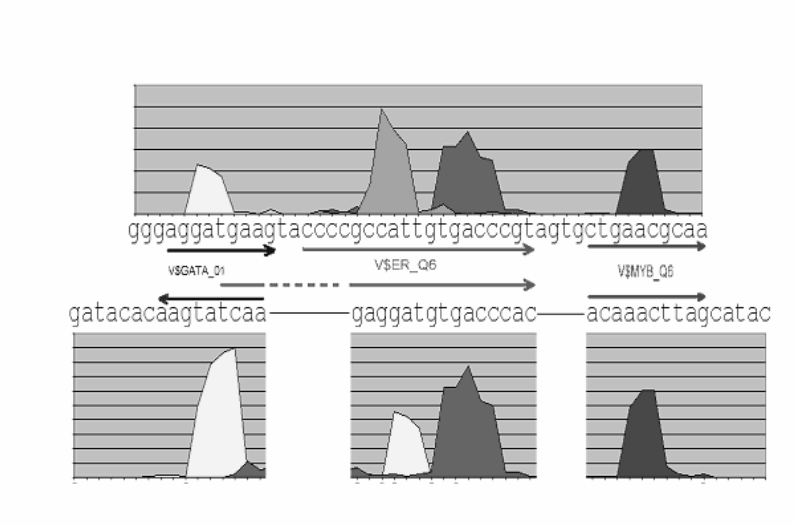


Рис. 3. Демонстрация работы алгоритма на примере двух сгенерированных последовательностей

Профили выравниваются друг с другом наряду с последовательностями. Стрелками обозначены потенциальные ССТФ и соответствующие им транскрипционные факторы.

2.4. Метод антифутпринта

При большой гомологии последовательностей имеет смысл рассматривать не сходные сайты, а различия. Считается вероятным, что эти различия легли в основу разницы между данными видами.

2.5. Метод анализа группы последовательностей

Определим композиционный модуль (КМ) как набор факторов с некоторыми параметрами (такими как вес матрицы). Зададим целевую функцию модуля F(S), характеризующую присутствие этого комплекса в последовательности S. Алгоритм получает на вход два набора последовательностей: анализируемый и фоновый. Далее выбирается комплекс, максимизирующий $R=(F_+ - F_-)/(\delta_+ + \delta_-)$, где F_+ и F_- средние, а δ_+ и δ_- — дисперсии распределения F на анализируемой и фоновой выборках соответственно [6].

2.6. Метод поиска цис-элементов на основе данных с паттернов экспрессии

Другой подход заключается в поиске цис-элементов на основе набора промоторов и соответствующих им значений, характеризующих уровни экспрессии. При этом ищутся, как и в предыдущем случае, композиционные модули. За целевую функцию R берется корреляция F и уровня экспрессии.

2.7. Метод поиска цис-элементов с учетом контекста

Предположим, что имеются две выборки последовательностей: позитивная $Q = \{q_1,...,q_m\}$ и негативная $T = \{t_1,...,t_k\}$. Позитивная выборка содержит последовательности, в которых присутствуют цис-элементы заданного типа, а негативная содержит последовательности, где таких цис-элементов нет. С помощью некоторого метода осуществим поиск сайтов и используем информацию о позитивной и негативной выборках для фильтрации сайтов. Введем правило $f(s) \in R$ такое, что если $f(s) > 0$, то s считается распознанным цис-элементом, иначе — не является. Зададим *f(s)* следующим образом: $f(s) = \sum_{i=0}^{N} f_i(s)$, где $f_i(s) = c_i^1$, если в районе $\left[p_i^1, p_i^2\right]$ присутствует последовательность (блок) s_i, и $f_i(s) = c_i^2$, если не присутст-

вует. Блок-моделью назовем тройку $< f_i, p_i^1, p_i^2 >$. Итак, по выборкам Q и T получаем блок-модели, а затем используем их при распознавании.

Для получения блок-моделей применим критерий максимального правдоподобия: $c_i^1 = \log(fr_i^1) - \log(fr_i^2)$, $c_i^2 = \log(1 - fr_i^1) - \log(1 - fr_i^2)$, где fr_i^1 — частота встречаемости блока s_i в районе $\left[p_i^1, p_i^2 \right]$ в выборке $Q = \{q_1, ..., q_m\}$, а fr_i^2 — частота встречаемости блока s_i в районе $\left[p_i^1, p_i^2 \right]$ в выборке $T = \{t_1, ..., t_k\}$. Далее выберем N моделей с наибольшей разностью $\left| c_i^1 - c_i^2 \right|$. По этим моделям будем проводить фильтрацию сайтов, найденных на произвольной последовательности. Если $f(s) > 0$, то s удовлетворяет фильтру, иначе — не удовлетворяет.

3. МОДУЛЬ СРАВНЕНИЯ МЕТОДОВ ПОИСКА

Качество распознавания может быть оценено распределением двух величин: ошибкой предсказания первого (*FP*) и второго (*FN*) родов. В зависимости от параметров поиска получим распределение этих ошибок. Введем следующие величины: значение предсказания $\alpha = 1 - FP$ и чувствительность $\beta = 1 - FN$. Пусть $S = \{s_1, ..., s_n\}$ — известные экспериментальные сайты, $Q = \{q_1, ..., q_m\}$ — сайты, найденные определенным методом. Обозначим $s_i \approx q_j$, это значит, что сайт s_i совпадает с сайтом q_j (распознан сайтом q_j). Пусть $Q' = \{q_j \in Q \mid \exists s_i \in S, s_i \approx q_j\}$ — правильно распознавшие сайты, $S' = \{s_i \in S \mid \exists q_j \in Q, s_i \approx q_j\}$. Тогда

$$\alpha = \frac{|Q'|}{|Q|}, \beta = \frac{|S'|}{|S|}.$$

Основная проблема состоит в том, что далеко не все сайты в геноме открыты и не все из открытых содержатся в соответствующих базах данных. Пусть $T = \{t_1, ..., t_k\}$ — неизвестные сайты. В объединении с известными неизвестные сайты дают все множество сайтов $S^* = S \cup T$. Перепишем значение предсказания и чувствительность с учетом неизвестных сайтов.

$$\alpha^* = \frac{|Q'^*|}{|Q|},\ \beta^* = \frac{|S'^*|}{|S^*|},$$

где $Q'^* = \{q_j \in Q \mid \exists s_i \in S^*, s_i \approx q_j\}$, $S'^* = \{s_i \in S^* \mid \exists q_j \in Q, s_i \approx q_j\}$. Можем записать, что $S'^* = S' \cup T'$ и $Q'^* = Q' \cup Q'_T$. Пусть неизвестных сайтов в k_T раз больше, чем известных $|T| = k_T |S|$. Пусть методы распознают меньший процент неизвестных сайтов, чем известных

$$\frac{|Q'_T|}{|Q'|} = k_\alpha \frac{|T|}{|S|} = k_\alpha \cdot k_T,\ k_\alpha \in (0,1].$$

Пусть также количество распознанных неизвестных сайтов зависит от количества распознанных известных сайтов, аналогично

$$\frac{|T'|}{|S'|} = k_\beta \frac{|T|}{|S|} = k_\beta \cdot k_T,\ k_\beta \in (0,1].$$

Тогда получим, что

$$\alpha^* = \alpha \cdot (1 + k_\alpha \cdot k_T),\ \beta^* = \beta \cdot \frac{1 + k_\beta \cdot k_T}{1 + k_T}.$$

Заметим, что в общем случае для различных транскрипционных факторов получаются различные константы. Константа k_T не зависит от исследуемого метода поиска. Так как сайты из множества T неизвестны, то предположим, что остальные константы k_α и k_β также не зависят от исследуемого метода. Тогда для сравнительного анализа методов достаточно использовать распределение $\langle \alpha, \beta \rangle$, имея в виду, что это не абсолютная, а относительная оценка методов. Качество метода распознавания варьируется для разных факторов, для разных групп последовательностей так же, как и для параметров метода. Параметры $\langle \alpha, \beta \rangle$ несравнимы для различных групп факторов и групп последовательностей, но сравнимы внутри одной группы факторов и последовательностей.

Сравнение реализовано в системе в виде модуля. Для добавления нового тестируемого метода достаточно реализовать функцию с использованием реализованных механизмов подсчета статистики. Если метод требует ис-

пользования дополнительных данных, то эти данные тоже должны быть добавлены таким образом, чтобы для подсчета статистики в методах использовался один и тот же набор генов и транскрипционных факторов. В процессе работы метода статистика сохраняется в виде, удобном для визуализации.

4. ОБЬЕКТНО-ОРИЕНТИРОВАННАЯ СИСТЕМА ПОИСКА ЦИС-ЭЛЕМЕНТОВ

Среда GRESA DT имеет иерархическую структуру. Вся функциональность разбита на классы, а классы сгруппированы в 3 основных пакета: ядро, набор общепринятых инструментов, набор экспериментальных инструментов.

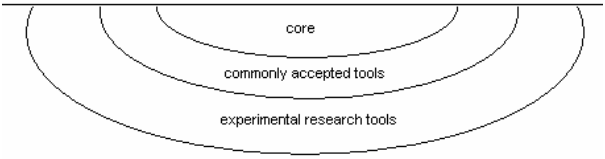


Рис. 4. Структура среды GRESA DT

Пакет «ядро» состоит из классов, представляющих основные общепринятые объекты биоинформатики регуляторных последовательностей ДНК.

Последовательность — последовательность ДНК. Представляет собой линейную последовательность нуклеотидов, обозначаемых буквами A,C,G,T. Также имеет название, описание и привязку к геному, т.е. номер хромосомы, стартовую позицию на хромосоме и направление “+” или “–”.

Сайт — подпоследовательность цепочки ДНК длиной, как правило, 10-20 нуклеотидов, имеющая позицию, длину, направление.

Фактор — объект, реализующий свойства транскрипционного фактора. Транскрипционный фактор — это белок, который связывается с сайтом на ДНК.

Выравнивание — несколько выровненных последовательностей. В каждой из них между нуклеотидами могут быть вставлены гэпы (промежутки). Выравнивание отражает эволюционное сходство последовательностей.

Набор последовательностей, сайтов, факторов — классы, в которых реализованы в основном сохранение и загрузка из общепринятых форматов, а также набор вспомогательных классов. Над объектами реализованы классические операции, такие как получение комплиментарной последовательности, поиск и др.

Набор общепринятых инструментов состоит из таких приложений, как MATCH, COMATCH (поиск композиционных модулей), footprint, CM SEARCH и др.

- MATCH — метод поиска сайтов на основе весовых матриц. Самый широко используемый в настоящее время метод.
- COMATCH — метод поиска композиционных элементов и сайтов с двумя доменами.
- FOOTPRINT — метод, учитывающий эволюционное сходство последовательностей. Вначале производится выравнивание последовательностей, а затем поиск сайтов, которые встретились на обеих последовательностях в одном и том же блоке выравнивания.
- CM SEARCH — метод поиска композиционных модулей, регулирующих группу генов. Для данной группы генов ищется общий модуль, предположительно регулирующий эти гены.

Набор экспериментальных инструментов состоит из еще не опубликованных приложений, находящихся в стадии разработки. Среди них наработки по поиску сайтов с использованием контекста, средства оценки качества распознавания методов.

Разработка и применение. Среда GRESA DT постоянно дополняется и развивается. Разработка среды по технологии Extreme Programming дает возможность постоянно поддерживать рабочую версию. Стабильность, при довольно большой и распределенной группе разработчиков, поддерживается за счет большого количества автоматизированных тестов. Жизненный цикл отдельного приложения состоит из этапов, когда приложение находится в стадии экспериментальной разработки, затем переходит в стабильную стадию.

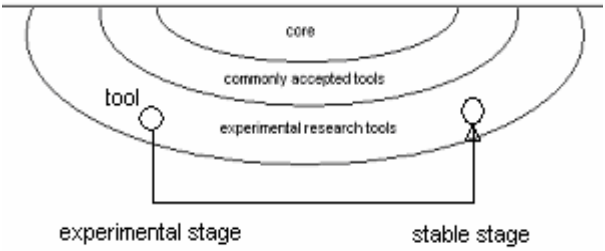


Рис. 5. Жизненный цикл отдельного приложения

Далее оно может перейти в набор общепринятых инструментов. Любой член команды может вносить изменения в любой класс, главное — сохранить успешное выполнение тестов.

На данный момент GRESA DT используется для обработки регуляторных ДНК последовательностей. Применение охватывает широкий круг задач распознавания сайтов. Также реализованы некоторые методы предсказания регуляции на основе предсказанных сайтов. Есть возможность построения комбинаций методов, например, как основу footprint-поиска сайтов можно взять либо результат match-поиска, либо результаты какого-либо другого метода, поддерживающего нужный формат записи. В GRESA DT также поддерживается сравнительное тестирование методов. Система тестирования оценивает качество распознавания сайтов. В данный момент в качестве выборки для тестирования используется база данных TRANSFAC [7].

Реализация и системные требования. Система реализована на языке C++ с использованием среды Microsoft Visual Studio. Операционная система Windows. Использованы технологии разработки ПО Extreme Programming и Microsoft Solution Framework. Системные требования зависят от задачи.

5. ЗАКЛЮЧЕНИЕ

Разработан и реализован инструментарий, позволяющий производить полноценный поиск цис-элементов, наиболее полно использующий данные, имеющиеся у экспериментатора. Каждый из методов может быть использован как в отдельности, так и в качестве дополнительного фильтра результа-

тов другого метода. Существует возможность простого и эффективного создания новых алгоритмов на базе уже существующих. Система эффективно используется в нескольких организациях.

СПИСОК ЛИТЕРАТУРЫ

1. Doolittle R. F. Microbial genomes opened up // Nature. — 1997. — Vol. 392. — P.339–342.
2. Maley L. E., Marshall C. R. The coming of age of molecular systematics // Science. — 1998. — Vol. 279. — P. 505–506.
3. Kel AE, Gossling E, Reuter I, Cheremushkin E, Kel-Margoulis OV, Wingender E. MATCH: A tool for searching transcription factor binding sites in DNA sequences // Nucleic Acids Res. — 2003. — Vol. 31, N 13. — P. 3576–3579.
4. Lawrence, C.E., Altschul, S.F., Bogouski, M.S., Liu, J.S., Neuwald, A.F., and Wooten, J.C. Detecting Subtle Sequence Signals: A Gibbs Sampling Strategy for Multiple Alignment // Science. — 1993. — Vol. 262. — P. 208–214.
5. Needleman SB, Wunsch CD. A general method applicable to the search for similarities in the amino acid sequence of two proteins // J. Mol. Biol. — 1970. — Vol. 48, N 3. — P.443–53
6. Kel-Margoulis OV, Ivanova TG, Wingender E, Kel AE. Automatic annotation of genomic regulatory sequences by searching for composite clusters // Pac. Symp. Biocomput. — 2002. — Vol. 7. — P. 187–198.
7. Wingender E., Chen X., Fricke E., Geffers R., Hehl R., Liebich I., Krull M., Matys V., Michael H., Ohnhäuser R., Prüß M., Schacherer F., Thiele S. and Urbach S. The TRANSFAC system on gene expression regulation // Nucleic Acids Res. — 2001. — Vol. 29. — P. 281–283.

СОДЕРЖАНИЕ

Предисловие редактора 5
Волянская Т. А. Международные стандарты представления в сети ИНТЕРНЕТ информационных ресурсов по культурному наследию: стандарт ANSI/NISO Z39.50 и профиль CIMI 7
Дортман П. А. Подходы к оптимизации программ в системе SFP 43
Дунаев А. А. Программный комплекс для исследования больших одномерных массивов данных с применением кратномасштабного анализа 50
Евстигнеев В. А. Многочлены Эрхарта 60
Касьянов В. Н., Касьянова Е. В. Дистанционное обучение: методы и средства адаптивной гипермедиа 80
Касьянов В. Н., Мирзуитова И. Л. Реструктурирующие преобразования: алгоритмы распараллеливания циклов 142
Касьянова Е. В. Язык программирования Zonnon для платформы .NET 189
Малинина Ю. В. Электронная среда коллективного накопления и каталогизации информации по преобразованиям программ 206
Маркин В. А., Маркина С. А. Система для быстрого прототипирования распараллеливающего компилятора ПРОГРЕСС-2. Ядро системы. Сценарий системы 217
Серебренников А. Л. Стандартные и новые подходы к архитектуре и методам обучения в среде Significo, основные направления развития среды 229
Черемушкин Е. С., Коновалова Т. Г., Мурзин Ф. А., Кель А. Э. Система распознавания цис-элементов на последовательностях ДНК 255

CONTENTS

Preface 5
Volyanskaya T. A. International standards of the Internet-representation of information related to cultural heritage: the ANSI/NISO Z39.50 standard and the CIMI Profile 7
Dortman P. A. Program optimization in SFP 43
Dunaev A. A. The software system for analysis of large linear data arrays with the use of the multiresolution wavelet analysis 50
Evstigneev V. A. Ehrhart polynomials 60
Kasyanov V. N., Kasyanova E. V. Distance education: methods and tools of adaptive hypermedia 80
Kasyanov V. N., Mirzuitova I. L. Restructuring transformations: loop parallelization algorithms 142
Kasyanova E. V. The programming language Zonnon for .NET Framework . 189
Malinina Yu. V. Electronic environment for collaborative accumulation and catalogization of information on program transformation 206
Markin V. A., Markina S. A. PROGRESS-2 — a system for fast prototyping of a parallelizing compiler. The system kernel. The system scenario .. 217
Serebrennikov A. L. Standard and new approaches to the architecture and methods of educating in the Significo environment, guidelines of development of environment 229
Cheremushkin E. S., Konovalova T. G., Murzin F. A., Kel A. E. A system of recognition of cis-elements on DNA sequences 255

УДК 519.68 + 681.3.06

Международные стандарты представления в сети ИНТЕРНЕТ информационных ресурсов по культурному наследию: стандарт ANSI/NISO Z39.50 и профиль CIMI / Волянская Т. А. // Программные средства и математические основы информатики. — Новосибирск, 2004. — С. 7–42.

Статья содержит краткий обзор стандарта ANSI/NISO Z39.50 и профиля CIMI. Стандарт Z39.50 определяет прикладную службу и спецификацию протокола для поиска и извлечения информации из баз данных. Z39.50 предназначен для унификации сетевого доступа к базам данных и определяет процедуры поиска, извлечения и форматы представления информации. Профиль CIMI служит спецификацией использования стандарта Z39.50 для доступа к информации о культурном наследии. В статье рассматриваются основы Z39.50, модели поиска и извлечения информации по Z39.50, определяются понятия абстрактной базы данных, пунктов доступа, схемы базы данных, абстрактной структуры записи, RPN-запросов, приводятся наборы поисковых атрибутов и наборы тэгов, определенные в Z39.50, введенные понятия иллюстрируются на примере. В статье дается краткий обзор профиля CIMI, рассматриваются спецификации поиска, выбора и передачи записей, приводятся набор атрибутов CIMI-1, набор тэгов CIMI Tag Set, CIMI схема и абстрактная структура записи, рассматриваются различные уровни семантической интероперабельности и приводится пример распределенной информационной системы на базе Z39.50. — Библиогр.: 11 назв.

International standards of the Internet-representation of information related to cultural heritage: the ANSI/NISO Z39.50 standard and the CIMI Profile / Volyanskaya T. A. // Software tools and mathematical foundations of informatics. — Novosibirsk, 2004. — P. 7–42.

The paper provides a brief review of the ANSI/NISO Z39.50 (ISO 23950) standard and the CIMI Profile. This standard Z39.50 describes the Application Service and specifies the Application Protocol for search and retrieval of information in databases. Z39.50 is intended for unification of network access to databases and specifies procedures and formats for a client to search and to retrieve database records. The CIMI Profile is a set of technical specifications for using ANSI/NISO Z39.50 to search and retrieve information related to our cultural heritage. Z39.50 basic services, information search and retrieval models are considered, the definitions of Abstract Database, Access Points, Database Schema, Abstract Re-cord Structure, RPN Query are determined, Attribute Sets and Tag Sets defined in Z39.50 are given, the definitions are illustrated by examples. The paper gives a brief review of the CIMI Profile. The database records search, selection and transfer specifications are considered; CIMI-1 Attribute Set, CIMI Tag Set, CIMI Schema and Abstract Record Structure are given; three levels of semantic interoperability are described; an example of Z39.50-based distributed information retrieval system is given.. — Refs: 11 titles.

УДК 519.68 + 681.3.06

Подходы к оптимизации программ в системе SFP / Дортман П. А. // Программные средства и математические основы информатики. — Новосибирск, 2004. — С. 43–49.

В настоящей работе рассматриваются подходы к осуществлению оптимизи-

рующих преобразований программ в системе функционального программирования SFP. Описаны алгоритмы для выполнения некоторых традиционных оптимизирующих программ, представленных в виде потоковых графов. — Библиогр.: 3 назв.

Program optimization in SFP / Dortman P. A. // Software tools and mathematical foundations of informatics. — Novosibirsk, 2004. — P. 43–49.

The paper describes the main approaches to implementation of optimizing transformations in the SFP programming system. The algorithms of some conventional optimizations are also given.. — Refs: 3 titles.

УДК 519.68 + 681.3.06

Программный комплекс для исследования больших одномерных массивов данных с применением кратномасштабного анализа / Дунаев А. А. // Программные средства и математические основы информатики. — Новосибирск, 2004. — С. 50–59.

При решении ряда научно-исследовательских и практических задач возникает проблема обработки больших массивов данных. Настоящая статья исследует способы организации работы с большими линейными массивами данных на примере обработки результатов кратномасштабного анализа нуклеотидной последовательности. Рассмотрены несколько стратегий предварительной выборки данных. Кроме того, описаны методы обработки нуклеотидных последовательностей и способы отображения результатов вычислений. — Библиогр.: 11 назв.

The software system for analysis of large linear data arrays with the use of the multiresolution wavelet analysis / Dunaev A. A. // Software tools and mathematical foundations of informatics. — Novosibirsk, 2004. — P. 50–59.

The problem of large data array processing often arises in various scientific research and applications. This paper illustrates the ways of working with large linear data arrays by the example of processing the results of multiresolution wavelet analysis of nucleotic sequences. Some strategies of data prefetching are described. In addition, the methods of processing the nucleotic sequences and displaying the results of such processing are considered. — Refs: 11 titles.

УДК 519.68 + 681.3.06

Многочлены Эрхарта / Евстигнеев В. А. // Программные средства и математические основы информатики. — Новосибирск, 2004. — С. 60–79.

В статье излагаются основы теории полиномов Эрхарта и её применения к некоторым проблемам, возникающим в NUMA-архитектурах. — Библиогр.: 3 назв.

Ehrhart polynomials / Evstigneev V. A. // Software tools and mathematical foundations of informatics. — Novosibirsk, 2004. — P. 60–79.

In this article, Ehrharts polynomials are introduced and several applicaitons to NUMA architectures are shown. — Refs: 3 titles.

УДК 519.68 + 681.3.06

Дистанционное обучение: методы и средства адаптивной гипермедиа / Касьянов В. Н., Касьянова Е. В. // Программные средства и математические основы информатики. — Новосибирск, 2004. — С. 80–141.

В последнее время адаптивные гипермедиа-системы становятся все более и более популярными в дистанционном обучении и предоставляют средства доступа к информации, управляемые пользователем. Адаптивные гипермедиа-системы сводят воедино идеи гипермедиа-систем и интеллектуальных обучающих систем и делают возможным персонализированный доступ к информации.

В данной работе рассматриваются вопросы поддержки дистанционного обучения, особое внимание уделяется анализу методов и средств адаптивной гипермедиа, используемых современными адаптивными обучающими Web-системами. — Библиогр.: 100 назв.

Distance education: methods and tools of adaptive hypermedia / Kasyanov V. N., Kasyanova E. V. // Software tools and mathematical foundations of informatics. — Novosibirsk, 2004. — P. 80–141.

Recently, hypermedia systems become more and more popular as tools for a user-driven access to information. Adaptive hypermedia systems bring together the ideas from hypermedia systems and intelligent tutoring systems, and enable personalized access to information.

In the paper the problems of distance education support are considered, most attention is given to analysis of methods and tools of adaptive hypermedia that are used by modern educational adaptive Web-systems. — Refs: 100 titles.

УДК 519.68 + 681.3.06

Реструктурирующие преобразования: алгоритмы распараллеливания циклов / Касьянов В. Н., Мирзуитова И. Л. // Программные средства и математические основы информатики. — Новосибирск, 2004. — С. 142–188.

Статья посвящена алгоритмам распараллеливания циклов — одного из наиболее эффективных реструктурирующих преобразований. Эти алгоритмы используют различные математические инструменты и различные представления зависимостей по данным. В статье мы приводим описание основных алгоритмов распараллеливания циклов и даем сопоставление их сильных и слабых сторон как на примерах, так и в сравнении “оптимальных” результатов. — Библиогр.: 66 назв.

Restructuring transformations: loop parallelization algorithms / Kasyanov V. N., Mirzuitova I. L. // NSoftware tools and mathematical foundations of informatics. — Novosibirsk, 2004. — P. 142–188.

The paper is devoted to loop parallelization algorithms — one of the most effective restructuring transformations. These algorithms make use of different mathematical tools and various representations of data dependences. In the paper we describe the main loop parallelization algorithms and assess their power and limits by using the examples, as well as by comparing the “optimal” results. — Refs: 66 titles.

УДК 519.68 + 681.3.06

Язык программирования Zonnon для платформы .NET / Касьянова Е. В. // Программные средства и математические основы информатики. — Новосибирск, 2004. — С. 189–205.

В статье кратко представлен новый язык программирования Zonnon, работа над которым ведется в институте информатики г. Цюриха. Разрабатываемый

язык задуман как современная альтернатива хорошо известному языку Оберон, являющемуся преемником языков Паскаль и Модула-2. — Библиогр.: 7 назв.

The programming language Zonnon for .NET Framework / Kasyanova E. V. // Software tools and mathematical foundations of informatics. — Novosibirsk, 2004. — P. 189–205.

The paper outlines a new programming language Zonnon being under development in the Institute of Computer Systems in Zurich. The language is aimed to be a modern alternative to a well-known Oberon language which is an evolution of Pascal and Modula-2 languages. — Refs: 7 titles.

УДК 519.68 + 681.3.06

Электронная среда коллективного накопления и каталогизации информации по преобразованиям программ / Малинина Ю. В. // Программные средства и математические основы информатики. — Новосибирск, 2004. — С. 206–216.

В данной работе описаны различные способы асинхронного взаимодействия, которые снимают проблемы коммуникаций и трудности, связанные с организацией семинаров, телефонных или электронных конференций. В настоящее время богатство и разнообразие асинхронных коммуникаций существенно выросло. Если раньше мы были ограничены текстовой формой электронных писем и бюллетеней, то теперь эта форма взаимодействия стала более гибкой.

Предлагаемая статья описывает опыт внедрения электронной среды для организации совместной работы на основе технологии WikiWiki. — Библиогр.: 6 назв.

Electronic environment for collaborative accumulation and c atalogization of information on program transformation / Malinina Yu. V. // Software tools and mathematical foundations of informatics. — Novosibirsk, 2004. — P. 206–216.

New approaches and technologies can greatly change the process of an interaction during a cooperative work. A special emphases is given to different ways of an anisochronous interaction which remove the problems of communication and difficulties connected with organization of meetings and telephone or electronic conferences. At present, the variety of means of anisochronous communication greatly increases. If earlier we were limited to the text form of electronic letters and bulletins, now this form of interaction becomes more flexible.

This paper describes our experience of deployment of the electronic groupware environment aimed to support collaboration on the basis of WikiWiki technology. — Refs: 6 titles.

УДК 519.68 + 681.3.06

Система для быстрого прототипирования распараллеливающего компилятора ПРОГРЕСС-2. Ядро системы. Сценарий системы / Маркин В. А., Маркина С. А. // Программные средства и математические основы информатики. — Новосибирск, 2004. — С. 217–228.

Статья описывает текущее состояние работ по созданию системы ПРОГРЕСС-2 для быстрого прототипирования распараллеливающего компилятора. Система создается как конструктор для построения прототипа компилятора,

кирпичиками которого являются различные функциональные и инструментальные компоненты. Особое внимание в статье уделяется ядру системы и средствам задания сценария работы создаваемого компилятора. — Библиогр.: 3 назв.

PROGRESS-2 — a system for fast prototyping of a parallelizing compiler. The system kernel. The system scenario / Markin V. A., Markina S. A. // Software tools and mathematical foundations of informatics. — Novosibirsk, 2004. — P. 217–228.

The paper describes the current state in constructing PROGRESS-2 — a system for fast prototyping of parallelizing compilers. The system is designed as a constructor for creating a compiler prototype whose blocks are different functional and instrumental components. Particular attention is given to the system kernel and tools for specifying a scenario of work of the compiler under construction. — Refs: 3 titles.

УДК 519.68 + 681.3.06

Стандартные и новые подходы к архитектуре и методам обучения в среде Significo, основные направления развития среды / Серебренников А. Л. // Программные средства и математические основы информатики. — Новосибирск, 2004. — С. 229–254.

В статье рассматриваются следующие вопросы: задачи, решаемые нейросетями, ожидаемые методы предобработки данных в среде Significo, описание стандартных и новых архитектур нейросетей с приведением результатов сравнительных тестов, рекурсивные и схоластические методы обучения нейросетей, усовершенствованный рекурсивный метод и сравнительное тестирование методов, направления и подходы в развитии интерфейсной части среды Significo. — Библиогр.: 6 назв.

Standard and new approaches to the architecture and methods of educating in the Significo environment, guidelines of development of environment / Serebrennikov A. L. // Software tools and mathematical foundations of informatics. — Novosibirsk, 2004. — P. 229–254.

The paper describes the following: the problems decided by neuronets; the methods of data pretreatment in the Significo environment; the description of standard and new architectures of neuronets supplied with the results of comparative tests; recursive and scholastic methods of teaching a neuronet; an advanced recursive method and comparative testing of methods; directions and approaches in the development of the interface part of the Significo environment. — Refs.: 6 titles.

УДК 519.68 + 681.3.06

Система распознавания цис-элементов на последовательностях ДНК / Черемушкин Е. С., Коновалова Т. Г., Мурзин Ф. А., Кель А. Э. // Программные средства и математические основы информатики. — Новосибирск, 2004. — С. 255–269.

Разработан и реализован инструментарий, позволяющий производить полноценный поиск цис-элементов, наиболее полно использующий данные, имеющиеся у экспериментатора.

Разработаны алгоритмы анализа больших районов ДНК на основе визуализации ДНК последовательностей.

Алгоритмы реализованы в объединенной объектно-ориентированной среде. Каждый из алгоритмов может быть использован как в отдельности, так и как дополнительный фильтр результатов другого алгоритма. За счет реализации существует возможность простого и эффективного создания новых алгоритмов на базе уже существующих. Система эффективно используется в нескольких организациях. — Библиогр.: 7 назв.

A system of recognition of cis-elements on DNA sequences / Cheremushkin E. S., Konovalova T. G., Murzin F. A., Kel A. E. // Software tools and mathematical foundations of informatics. — Novosibirsk, 2004. — P. 255–269.

A set of computational tools has been developed for a complete search of cis-elements with the maximum use of available biological data.

A wide range of DNA analysis algorithms based on visualization of genetic information have been developed.

The algorithms have been implemented in C++ in a common object-oriented environment. Every algorithm can be used separately and as an additional filter for another algorithm's results. Implementation allows a new algorithm to be easily developed on the basis of already existing ones. The system was successfully used in several organizations. — Refs: 7 titles.

ПРОГРАММНЫЕ СРЕДСТВА И МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

Под редакцией
проф. Виктора Николаевича Касьянова

Рукопись поступила в редакцию 15. 02. 2004
Ответственный за выпуск Г. П. Несговорова
Редактор З. В. Скок

Подписано в печать 12. 08. 2004
Формат бумаги 60 × 84 1/16 Объем 15,9 уч.-изд.л., 17,4 п.л.
Тираж 75 экз.

ЗАО РИЦ "Прайс-курьер" 630090, г. Новосибирск, пр. Акад. Лаврентьева, 6,
тел. (383-2) 34-22-02