СОВРЕМЕННЫЕ ПРОБЛЕМЫ КОНСТРУИРОВАНИЯ ПРОГРАММ

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***

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

СОВРЕМЕННЫЕ ПРОБЛЕМЫ КОНСТРУИРОВАНИЯ ПРОГРАММ

Под редакцией
проф. Виктора Николаевича Касьянова

Новосибирск 2002

УДК 519.68 + 681.3.06
ББК 22.183.49

Современные проблемы конструирования программ. — Новосибирск: Ин-т систем информатики им. А. П. Ершова СО РАН, 2002. — 256 с.

Сборник является девятым в серии книг, издаваемых Институтом систем информатики им. А. П. Ершова СО РАН по проблемам конструирования и оптимизации программ. Он посвящен решению актуальных задач конструирования эффективных и надежных программ и систем на основе теоретико-графовых методов, функционального программирования, сетевой обработки и средств визуализации.

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

MODERN PROBLEMS OF PROGRAM CONSTRUCTION

Edited by
prof. V. N. Kasyanov

Novosibirsk 2002

This volume is the ninth one in a series of books published in A. P. Ershov Institute of Informatics Systems on the problems of program construction and optimization. It describes solutions to actual problems of constructing efficient and reliable programs and software systems based on graph-theory methods, functional programming, network processing and visualizing tools.

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

ПРЕДИСЛОВИЕ РЕДАКТОРА

Девятый выпуск серии "Конструирование и оптимизация программ" посвящен решению актуальных задач разработки методов и средств конструирования эффективных и надежных программ. Продолжая уже сложившиеся традиции, данный выпуск, как и предыдущие, базируется на результатах исследований, выполненных в лаборатории по конструированию и оптимизации программ ИСИ СО РАН совместно с НГУ при финансовой поддержке РФФИ и Минобразования.

В прошедшем году сибирская школа информатики и программирования понесла тяжелую утрату. От нас ушли два ее ведущих профессора: Игорь Васильевич Поттосин и Майя Михайловна Бежанова, и мы посвящаем наш скромный труд светлой их памяти.

Более четверти века основной курс по программированию студентам мехмата НГУ читался нами с Игорем Васильевичем на разных потоках в параллель. Когда на мехмате был образован еще и третий поток, лекции на нем вслед за мной и проф. В.К. Сабельфельдом в последние годы стала читать Майя Михайловна. Отнесясь к курсу с душой и весьма ответственно, Майя Михайловна инициировала создание книги[1] по курсу, в подготовке которой я, к сожалению, не смог участвовать, поскольку завершал работу над задачником[2]. Судьбе было угодно сделать так, чтобы авторы так и не увидели результаты своего труда. Работа над книгой завершалась в декабре 2000, в последние дни жизни Майи Михайловны, а книга поступила в Новосибирск в декабре 2001, в день похорон Игоря Васильевича.

Я знаю Игоря Васильевича более 30 лет, еще со студенчества, с конца 60-х, был первым аспирантом, который защитился под его научным руководством, и одним из постоянных его соавторов. Вспоминаю 1971 г., год окончания НГУ, когда мы, молодые теоретики, я, Марк Трахтеброт и Виктор Сабельфельд, пришли по распределению в лабораторию Поттосина. А тогда в отделе Ершова было две программистских лаборатории: лаборатория Игоря Васильевича — лаборатория по системному программированию и лаборатория по теоретическому программированию, которую возглавлял

---

[1] М.М. Бежанова, И.В. Поттосин. Современные понятия и методы программирования. – М.: Научный мир, 2000.

[2] В.Н. Касьянов. Курс программирования на Паскале в заданиях и упражнениях. – Новосибирск: НГУ, 2001.

сам Андрей Петрович. Первое, что мы услышали от Игоря Васильевича, было то, что теория и практика в программировании развиваются скорее параллельно и что наша общая задача сделать это развитие совместным, взаимно обогащаемым. Это было его кредо. Объединение теоретических исследований с программными разработками – одна из основных черт сибирской школы информатики и программирования.

Игорь Васильевич был одним из основателей этой школы. Живой человек, которому ничего человеческое не было чуждо, он был настоящим русским интеллигентом и программистом от бога. В начале 90-х, когда для всех нас настали тяжелые времена, когда под вопросом стояло само существование нашего коллектива, он взял лидерство на себя и сохранил коллектив и институт. В последние годы, отказавшись от директорства, но сохранив неформальное лидерство и полную ответственность за судьбу нашей школы, он много работал, причем не только с коллегами и студентами, но и со школьниками, был в гуще всех событий у нас в стране и за рубежом. Смерть прервала эти работы, и нам, его ученикам и коллегам, выпала обязанность и честь быть продолжателями его дела по сохранению и развитию сибирской школы информатики и программирования.

23 января 2002 г. *Проф. В.Н. Касьянов*

Д. Е. Бабурин

ИЕРАРХИЧЕСКИЙ ПОДХОД ДЛЯ АВТОМАТИЧЕСКОГО РАЗМЕЩЕНИЯ АЦИКЛИЧЕСКИХ ГРАФОВ*

1. ВВЕДЕНИЕ

В настоящее время графовое представление информации все больше и больше используется в различных областях точных и естественных наук. В программировании графовые модели применяются при создании программного обеспечения (управляющие графы, иерархии классов, диаграммы потоков данных), дизайне баз данных (диаграммы сущностей-связей), разработке информационных систем и систем реального времени (модели компьютерных сетей, графы состояний, сети Петри), а также в системном программировании — при создании теории компиляции и преобразования программ. Вопрос визуализации подобных графовых структур является краеугольным камнем в процессе адекватного отображения информации [2, 8, 55].

Автоматическое размещение графов — это задача конструирования геометрического представления графа, удовлетворяющего набору эстетических критериев. В последние пятнадцать лет эта проблема подвергается интенсивному изучению. За это время было опубликовано большое количество работ, начиная с 92-го года проводятся ежегодные конференции. Тем не менее задача автоматического размещения графов так и остается до конца не решенной, в том смысле, что до сих пор не было предложено универсального метода ее решения.

Возникшую ситуацию можно объяснить тем, что алгоритм размещения должен «учитывать» такие разносторонние аспекты, как: класс графа, семантика графовой информации, соглашения об изображении, эстетические критерии и ограничения на получаемое изображение. Перечисленные аспекты настолько многогранны, и их выбор настолько сильно зависит от случая конкретного применения, что разработка универсального алгоритма на данном этапе не представляется возможной.

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

В связи с данной проблемой был разработан ряд подходов к автоматическому размещению графов, которые базируются на фиксации некоторого набора из указанных критериев/аспектов. Так, например, был разработан целый ряд методов, позволяющих разместить планарный граф с прямолинейными или ортогональными дугами, или методы, основанные на физической упруго-силовой модели, которые больше подходят для сильно связанных неориентированных графов с прямолинейными ребрами.

Далее мы сконцентрируем свое внимание на ациклических графах. Выбор нами этого подкласса можно объяснить двумя причинами. Во-первых, преимущественное большинство реальных графов, встречающихся в программировании, являются ациклическими, а во-вторых, любой ориентированный (и тем более неориентированный) граф может быть преобразован к ациклическому путем смены или задания ориентации у части его ребер. Подобные преобразования будут нами подробно обсуждены в пятом разделе данного обзора.

При размещении ациклических графов во внимание принимаются следующие эстетические критерии:

1) совпадающее направление ребер;
2) минимизация площади изображения;
3) равномерность распределения вершин по площади всего изображения;
4) отсутствие слишком длинных ребер;
5) минимизация количества пересечения ребер;
6) прямолинейность ребер.

Одновременная оптимизация всех вышеперечисленных критериев является невозможной потому, что эти критерии зачастую оказываются противоречащими друг другу. Так, например, условие прямолинейности ребер нарушает равномерность распределения вершин и увеличивает общую площадь изображения.

Существует несколько различных техник размещения ациклических графов. Каждая из этих техник ставит во главу угла один или несколько из вышеперечисленных эстетических критериев и старается в меру возможностей удовлетворить оставшиеся критерии.

С одной стороны, можно решать задачу размещения ациклических графов сведением их к планарному графу и дальнейшему построению выпуклого изображения, используя особенности топологии планарных графов, с последующим восстановлением изначальной структуры. Так, например, хорошо изучена задача построения выпуклого изображения для планарных ациклических *st-графов*, т.е. графов, имеющих одну входную и одну вы-

ходную вершины, а также планарную укладку, в которой дуга, соединяющая эти вершины, лежит во внешней грани. Они допускают монотонное изображение с прямолинейными ребрами, внешняя грань которого есть заданный треугольник. Изображение в этом случае строится рекурсивно, путем выделения специальным образом st-подграфов с меньшим множеством вершин, чем у исходного, и применения алгоритма к ним (см. например [1, 18]). При таком подходе, когда фиксируется внешняя грань и дуги изображены прямыми линиями, с возрастанием степеней вершин углы между смежными ребрами быстро уменьшаются, что сильно понижает читаемость изображения.

В [28] описан метод сведения произвольного планарного ациклического графа к st-графу с сохранением планарности, однако само требование планарности исходного графа является слишком жестким. Задача планаризации произвольного ациклического графа является NP-трудной [46] (под планаризацией мы здесь понимаем задачу удаления минимального количества ребер так, чтобы сделать граф планарным). Даже несмотря на обилие существующих эвристик для решения этой задачи, планаризация не слишком подходит нашим целям, поскольку нарушает саму структуру исходного графа.

Задачу автоматического размещения ациклических графов можно также решать с помощью ортогональных техник, которые широко используются для графов произвольного вида и имеют хорошие временные показатели [12]. В этом случае при рассмотрении графов степени вершин более 4-х возникает проблема с появлением разброса в размерах вершин. Дело в том, что вершины с большой степенью размещают свои ребра за счет увеличения размеров, это приводит к одновременному наличию и слишком больших, и совсем маленьких вершин. Здесь главным недостатком, с нашей точки зрения, является то, что мы теряем возможность отражения иерархичности графа, к тому же размещения, полученные с использованием ортогонального подхода, более разряжены, чем, например, планарные размещения, и, соответственно, имеют большую площадь, что понижает удобность изучения структуры графа.

Подход, описанный в [72], позволяет визуализировать произвольный ациклический граф методом его декомпозиции на кланы — подграфы, состоящие из имеющих общие связи с остальной частью графа вершин. Такой метод хорошо отражает структуру графа, но имеет неудовлетворительное качество проведения ребер. Изначально прямолинейные ребра могут пересекать вершины графа, а эффективность введенных эвристик для устранения таких случаев путем создания дополнительных изломов ребер тео-

ретически не обоснована. Похожий подход, основанный на принципе «разделяй и властвуй», описан в [60]. Детальный обзор по вопросам кластеризации и размещения иерархических графовых структур может быть найден в [7].

Наиболее же известную идею размещения ациклических графов можно рассматривать как обобщение случая размещения деревьев. Для подчеркивания иерархичности структуры используются, как и в древесном случае, поуровневые представления, в которых все дуги следуют одному и тому же направлению [66]. Такие представления называются монотонными поуровневыми представлениями.

Настоящая работа посвящена обзору такой методологии (иерархический подход), применяемой для размещения ациклических графов. Данный подход является крайне интуитивным и позволяет для произвольного ациклического графа построить монотонное поуровневое представление с ребрами в виде ломанных или сплайнов. Первоначально идея была сформулирована еще в 1981 году [74], а также в некоторых других очень близких по тематике работах [77, 20]. Дальнейшее развитие (см. аннотированную библиографию [9]) сохранило основную концепцию, в которой иерархический подход содержит три части:

1) распределение вершин по уровням так, чтобы все дуги следовали одному направлению;
2) выбор порядка вершин на уровне с целью минимизации пересечений ребер;
3) определение координат вершин на уровне с целью минимизации общей длины ребер и количества изломов.

Следует отметить, что данный подход оперирует различными эстетическими критериями на каждом из своих шагов.

Для успешного применения иерархического подхода система изображения графов (исходные данные, визуализирующая компонента, область применения) должна удовлетворять следующим ограничениям:

1) исходный граф должен быть ациклическим мультиграфом или мультиграфом, «близким» к ациклическому. Близость здесь следует понимать в том смысле, что смена ориентации малого количества дуг должна превратить граф в ациклический;
2) семантика графовой информации должна предполагать возможность построения монотонного изображения, подчеркивающего направленность или наличие некоторого порядка на множестве вершин;

3) система визуализации не должна предполагать проведение только прямолинейных ребер между вершинами графа. Ребра должны быть изображены в виде ломанных или сплайнов.

В следующем разделе нашего обзора приведено общее описание шагов иерархического подхода. Далее каждый их этих шагов подробно описан в отдельном разделе. Разд. 5 посвящен вопросам предварительного преобразования структуры графа. Общая структура и стиль данного обзора соответствует [10] и [14].

2. ЭТАПЫ ИЕРАРХИЧЕСКОГО ПОДХОДА

Канонически метод состоит из трех последовательно применяющихся шагов, которые изображены на рис. 1.

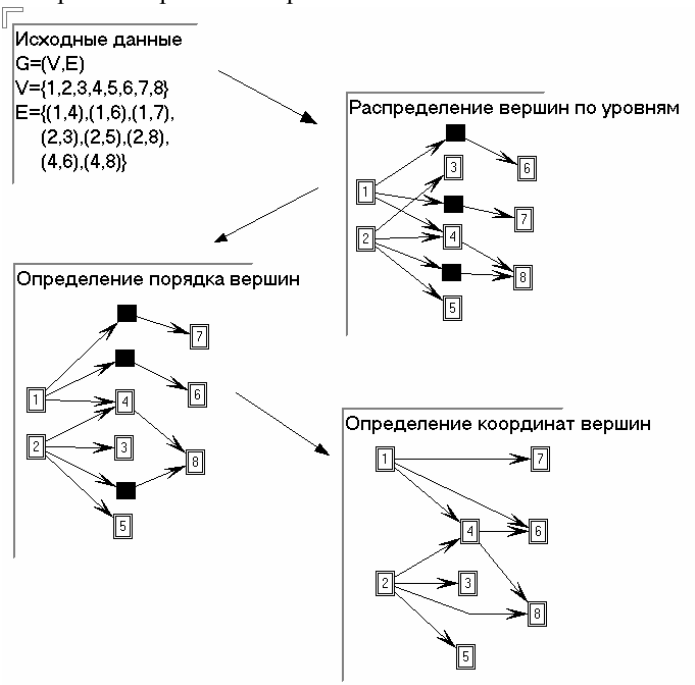


Рис. 1. Этапы иерархического подхода

- Распределение вершин по уровням. Каждой вершине u присваивается число L(u), указывающее уровень этой вершины так, чтобы все дуги, соединяющие вершины, следовали от меньшего номера к большему, при этом вершины одного уровня не должны быть связаны между собой. Подразумевается, что все вершины одного уровня будут расположены на одной прямой — вертикальной или горизонтальной — в зависимости от того, какое мы предпочитаем общее направление размещения: сверху вниз или слева направо. После проведения распределения вершин по уровням просматриваются все дуги: если дуга e = (u,v) проходит через несколько уровней, т.е. L(u) – L(v) > 1, то она удаляется из графа и заменяется на цепочку *фиктивных* вершин $v_1,\ldots,v_N$ такую, что $v_1 = u$, $v_N = u$, каждая вершина v_i соединена дугой с v_{i+1} и $L(v_{i+1}) = L(v_i) + 1$, а $v_1 = v$ и $v_N = u$.
- Определение порядка вершин на уровне. Задача данного шага состоит в сортировке вершин на каждом из уровней. Порядок вершин определяет топологию конечного изображения и должен быть выбран с целью минимизации пересечений ребер графа.
- Определение координат вершин на уровне. На данном шаге каждой вершине присваивается вертикальная координата (при размещении в горизонтальном направлении) так, чтобы сохранить порядок вершин на уровне, который был вычислен на предыдущем шаге. Координата выбирается с целью минимизации общей длины ребер и количества изломов. Наконец, конечное изображение получается представлением каждого ребра в виде прямолинейного отрезка и удаления фиктивных вершин. В этом случае длинные ребра оказываются изображенными в виде ломанных.

Некоторые исследователи склонны выделять задачу проведения ребер в отдельный этап алгоритма. В этих случаях ребра изображаются не каноническими ломанными, а, например, сплайнами [40, 67]. Для проведения сплайнов, не пересекающих вершины графа и проходящих через заданные области плоскости, существует алгоритм, описанный в [27].

Также задача проведения ребер выделяется в отдельный этап, если вершины графа имеют неточечное представление [68]. При этом приходится следить за отсутствием пересечений ребер и вершин графа, а также определять точки на изображении вершины, в которые будут входить и выходить ребра, инцидентные этой вершине графа.

Как уже было сказано выше, в случае, когда исходный граф не является ациклическим, требуется отдельный шаг алгоритма, который отвечает за

препроцессирование графа и преобразование его к ациклическому ориентированному графу. После построения изображения структура изначального графа восстанавливается.

Проблема оптимальной расстановки меток на дугах графа применительно к иерархическому размещению рассмотрена в [51].

Иерархический подход оказывается не очень удачным с точки зрения построения инкрементального размещения. Этот факт можно объяснить тем, что топология получаемого изображения очень сильно зависит от выделенной на первом шаге алгоритма иерархии вершин. А даже при локальных изменениях графа, как-то удаление или добавление вершин, групп вершин или ребер, не удается сохранить эту иерархию неизменной. Несмотря на это, в работах [16, 63, 22] рассмотрены разные подходы к инкрементальному размещению ациклических графов.

Вопросы инкрементального размещения очень тесно переплетены с вопросом удовлетворения ограничений на получаемое изображение. В качестве ограничений, которые могут быть удовлетворены при таком подходе, можно назвать требование разместить две выделенные вершины с одного уровня иерархии как можно ближе друг к другу, а также выравнивание вертикальных координат вершин с разных уровней иерархии [16, 45].

Иерархический подход размещает граф по уровням, образуя монотонное изображение. Направление следования уровней может быть как горизонтальным, так и вертикальным. В реальных системах это должно определяться семантикой изображаемой информации. Так, например, для иерархий классов более привычной является вертикальная укладка, для графов вызовов — горизонтальная. Иногда в данном вопросе решающее значение играет визуализирующая компонента или какие-либо ограничения системы. При горизонтальной укладке преимущественным направлением дуг является горизонтальное, что позволяет размещать на них длинные надписи. Поскольку данный обзор не затрагивает вышеприведенных вопросов, то будем считать, что уровни размещаются слева направо.

3. РАСПРЕДЕЛЕНИЕ ВЕРШИН ПО УРОВНЯМ

Задачей данного шага является присваивание каждой вершине ее конечной горизонтальной координаты. Для этого исходный ациклический граф $G = (V, E)$ должен быть приведен к *поуровневому* ациклическому графу. Поуровневое разбиение есть разделение V на подмножества $L_1, L_2, \ldots, L_h$, так что для каждого ребра $(u, v) \in E$, где $u \in L_i$ и $v \in L_j$, верно то, что i

> j. *Высотой* разбиения будем называть количество уровней *h*, а *шириной* w — число вершин на самом большом уровне. *Зазором* ребра (u, v), где $u \in L_i$ и $v \in L_j$, назовем разницу i – j. Разбиение называется *сжатым*, если не существует ребра с зазором, большим единицы.

После разбиения вершин по уровням всем вершинам одного уровня присваивается одна горизонтальная координата, т.е. $V_x = i$ для всех вершин с уровня L_i.

3.1. Техника фиктивных вершин

Рис. 2. Пример графа, не имеющего сжатого разбиения

Существование поуровневого распределения для ациклических графов очевидно. Однако не для всякого ациклического графа существует его сжатое разбиение — это может быть продемонстрировано уже для графа, состоящего всего из трех вершин (рис. 2). Необходимость же построения сжатого разбиения диктуется следующими двумя причинами. Во-первых, сжатое разбиение минимизирует горизонтальную длину ребер и количество их изломов. При сжатом разбиении ребра связывают вершины только соседних уровней и могут быть изображены прямолинейными отрезками. Во-вторых, сжатое разбиение существенно упрощает задачу минимизации пересечений ребер графа, которая решается на втором шаге алгоритма. Сжатое разбиение сводит эту глобальную задачу к задаче минимизации пересечений для двух соседних уровней.

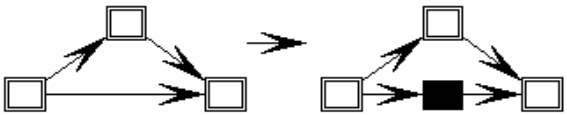


Рис. 3. Вставка фиктивных вершин

Для построения сжатого разбиения произвольного ациклического графа применяется техника *фиктивных* вершин. Фиктивные вершины добавляются в граф вдоль длинных ребер, т.е. ребер с зазором, большим единицы (рис. 3). Для каждого такого ребра (u, v), $u \in L_i$ и $v \in L_j$, с зазором $k = i - j > 1$ добавляется k – 1 вершин $u_1, \ldots u_{k-1}$ таких, что $u_m \in L_{i-m}$.

Такое построение приводит к тому, что в графе остаются только ребра, соединяющие вершины соседних уровней, и задача минимизации пересечения ребер локализуется до двухуровневого случая. Данное построение за-

одно решает и задачу проведения ребер в конечном изображении. Ребро изображается в виде ломанной, концами которой являются сами вершины, а точками излома — вставленные на первом этапе фиктивные вершины. Таким образом исчезает возможность пересечения ребер с вершинами.

Заметим здесь, что существуют работы по размещению ациклических графов, находящиеся в рамках данной методологии, однако не использующие технику фиктивных вершин [4]; в этих случаях дополнительное внимание уделяется вопросу проведения ребер графа.

3.2. Критерии построения разбиения

Постараемся сформулировать набор критериев, которые стоит принимать во внимание при построении поуровневого разбиения.

Искомое размещение графа должно быть компактно. Поскольку площадь фактически зависит только от количества уровней и максимального количества вершин на одном уровне, то компактизации можно добиться уменьшением одной из этих величин. При этом высота графа оказывается ограниченной снизу длиной максимального пути в графе. Существуют простые методы, позволяющие достичь нижней оценки высоты разбиения (п. 3.3), однако минимизация ширины есть NP-трудная задача, которая в реальных системах может быть решена только приближенно (п. 3.4). В связи с этим различием необходимо заметить, что предпочтение выбора величины для минимизации зачастую определяется на уровне визуализирующей системы. Если вершины графа изображаются не просто точками, а областями на плоскости, причем имеющими некоторые текстовые надписи, то при горизонтальной укладке наиболее важной становится высота разбиения, а при вертикальной — ширина. Это может быть еще одним хорошим доводом для использования горизонтального следования уровней в изображении графа.

Следующий важный критерий к разбиению вершин — это минимизация количества фиктивных вершин. Такое желание связано с тем, что введение фиктивных вершин повышает общую сложность алгоритма. Если мы рассматриваем достаточно плотный граф, в котором длинных ребер $O(N)$, то после их удаления мы получим $O(N^2)$ фиктивных вершин. Верхняя оценка числа вставленных фиктивных вершин может быть найдена в [36]. Немаловажно также и то, что чем больше мы вводим фиктивных вершин, тем больше получится в конечном размещении изломов на дугах. Таким образом, кроме минимизации общего числа вершин необходимо минимизировать максимальную длину ребра, то есть число фиктивных вершин, заме-

няющих одну дугу, потому как чем длиннее ребро и чем больше на нем изломов, тем сложнее при изучении изображения проследить соответствующую связь в графе. Ну и поскольку сложность всех шагов алгоритма размещения графа зависит от общего количества вершин, то мы должны стремиться к тому, чтобы не слишком увеличить это количество при добавлении фиктивных вершин. Способ построения разбиения с минимизацией количества фиктивных вершин рассмотрен в п. 3.5. К сожалению, одновременная минимизация высоты разбиения и количества фиктивных вершин уже оказывается NP-трудной задачей [53].

Поуровневое размещение иногда должно учитывать ограничения, налагаемые семантикой исходного графа. Так, в некоторых приложениях часть вершин оказывается предраспределенной по уровням. Это предраспределение может быть вызвано желанием выделить некоторые вершины или явно указать последовательность появления вершин (time-lines). Чаще всего такая задача может быть сведена к исходной путем предварительного слияния вершин с одинаковым предопределенным рангом.

3.2. Распределение по длиннейшему пути

Данный способ позволяет за линейное время распределить вершины графа по уровням так, чтобы высота получающегося разбиения оказалась минимальной. Для этого надо поместить все *стоки* (вершины без выходящих дуг) на уровень L_1, а каждую оставшуюся вершину — на уровень L_{p+1}, где p — длина наибольшего пути от этой вершины до ближайшего стока.

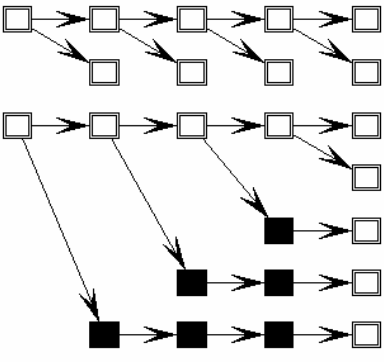


Рис. 4. Способы распределения вершин по уровням

Здесь следует отметить, что данный способ построения разбиения является инвариантным относительно направления дуг. При построении разбиения от стоков мы получим граф со всеми стоками, расположенными на одном уровне, но аналогичным образом разбиение можно построить и от *источников* (вершин без входящих дуг), и при этом все источники будут расположены на одном уровне. Очевидно, что высота обоих разбиений будет одинакова, однако ширина может отличаться на сколь угодно большую величину (рис.

4). Поскольку данный алгоритм является линейным, то представляется разумным построить оба разбиения и выбрать из них наименее широкое.

Основной недостаток распределения по длиннейшему пути — это отсутствие оптимизации на ширину разбиения. Проблемам минимизации ширины разбиения посвящен следующий раздел.

3.4. Минимизация ширины разбиения

К сожалению, задача нахождения разбиения с минимальной шириной и при этом ограниченной высотой является NP-полной [38]. NP-полнота задачи следует из сведения этой задачи к известной NP-полной задаче составления расписания для мультипроцессора. Каждая вершина ациклического графа представляет работу, которая может быть выполнена на любом из процессоров за единичное время. Ребро графа представляет ограничение предшествования на выполняющиеся работы. Составление расписания для мультипроцессора есть распределение работ по W процессорам так, чтобы все они могли быть выполнены за время, не превосходящее H. Понятно, что такое расписание существует только в том случае, если есть разбиение графа по уровням с высотой, меньшей H, и шириной, меньшей W.

Эквивалентность задачи поуровневого разбиения задаче составления расписания для мультипроцессора позволяет сразу же воспользоваться результатами, достигнутыми в этой области. Существует ряд эвристик для решения задачи составления расписания, и эти эвристики могут быть применены к нашей задаче. Например, эвристика Кофман-Графама [24] позволяет построить разбиение ширины W, и при этом его высота не будет превосходить $(2-2/W)h_{min}$, где h_{min} — минимально возможная высота разбиения шириной W. Данная оценка на высоту разбиения получена в [54].

Стоит заметить, что минимизируемая данными способами ширина разбиения может оказаться все равно не удовлетворительной, поскольку алгоритм не учитывает последующую вставку фиктивных вершин. В полученном графе могут оказаться уровни с огромным количеством ребер, идущих через них, и эти ребра будут заменены фиктивными вершинами, что расширит такие уровни. Однако, учитывая то, что в конечном изображении графа фиктивные вершины будут обратно заменены ребрами, можно сказать, что данный способ построения разбиения является вполне приемлемым, если ширина дуг графа мала по сравнению с размерами вершин. В противном случае предлагаются модификации эвристики Кофман-Графама, учитывающие ширину, «приносимую» фиктивным вершинами.

3.5. Минимизация количества фиктивных вершин

Существует полиномиальный алгоритм, позволяющий построить разбиение с минимальным количеством фиктивных вершин [40]. Нахождение оптимального разбиения формулируется в виде задачи линейного целочисленного программирования: $Min \sum_{(u,v)\in E} L_U - L_V$, где (u, v) — ребро с ограничениями: $L_U - L_V \geq 1$ и $L_U, L_V \geq 1$.

Существует ряд способов решения данной задачи за полиномиальное время. Первый — решить эквивалентную линейную задачу, а затем за полиномиальное время трансформировать решение в целочисленное. Другой способ — это сведение данной задачи к задаче минимизации потоков, для которой существуют полиномиальные решения [42].

Поскольку в такой постановке матрица ограничений является унимодулярной, то для решения может быть применен симплекс-метод, который хотя и не полиномиален, но на практике является быстросходящимся [40].

4. ОПРЕДЕЛЕНИЕ ПОРЯДКА ВЕРШИН НА УРОВНЕ

После построения поуровневого разбиения встает задача определения порядка расположения вершин на каждом уровне нашего графа. Нахождение такого распределения с целью минимизации количества пересечения ребер и есть задача данного этапа алгоритма.

В первую очередь стоит отметить, что количество пересечения ребер в поуровневом графе не зависит от конечных координат вершин, а зависит только от их относительного положения внутри каждого уровня. Таким образом, задача данного этапа является не геометрической, а всего лишь комбинаторной. Однако эта задача является NP-полной уже для графа, имеющего всего лишь два уровня [39], и, более того, NP-полной эта задача остается, даже если есть всего лишь одна вершина со степенью, большей единицы на каждом из уровней [59].

Задачу минимизации пересечений ребер, как в случае двух уровней [48, 76], так и глобальную [47, 44], естественно сформулировать в виде задачи целочисленного программирования. Однако, несмотря на возможность получения точного решения, такой подход может быть применен только для очень маленьких графов. Практические реализации показывают, что в разумное время решение задачи может быть найдено лишь для графов, содержащих не более 15 вершин (в случае глобальной оптимизации) и не бо-

лее 60 вершин для каждого из уровней (в случае минимизации пересечений лишь для двух соседних уровней).

В абсолютном большинстве известных автору работ, проблема минимизации пересечений ребер в поуровневом графе решается не глобально для всего графа, а лишь локально — для всех пар соседних уровней. Эксперименты [49] показывают, что сведение задачи минимизации пересечений ребер к случаю двух уровней дает результаты, очень далекие от оптимальных. Более приемлимых результатов минимизации глобального количества пересечений добиваются путем многократного «просматривания» всех соседних уровней и минимизации пересечений на них. Такая эвристика имеет несколько модификаций, которые различаются по следующим критериям:

- выбор способа фиксации уровней. При рассмотрении двух смежных уровней положение вершин на одном из уровней можно считать фиксированным и выбирать оптимальное расположение вершин другого уровня. Возможно также фиксирование положения вершин уровней L_{i-1}, L_{i+1} при рассмотрении уровня L_i;
- выбор порядка, в котором просматриваются пары соседних уровней. Уровни могут просматриваться слева направо или наоборот, возможно также чередование направлений просмотра;
- выбор критериев окончания просмотров. Это может быть фиксированное количество просмотров или же просмотры делаются, пока они уменьшают или существенно уменьшают общее количество пересечений.

Отдельно стоит поднять вопрос о выборе первоначального порядка вершин на каждом из уровней. Традиционно для этого применяется метод [40], позволяющий избежать пересечения ребер в случае, если исходный граф был деревом. Метод состоит в проведении поиска в глубину, при котором вершины получают номера на своем уровне в порядке их просмотра в ходе такого поиска.

Итак, в дальнейшем мы будем рассматривать только задачу минимизации количества пересечений для двух уровней. Пусть мы имеем двудольный граф $G = (L_1, L_2, E)$, где L_1 — множество вершин первого уровня, L_2 — вершины второго уровня, E — множество ребер. На основе фиксированного расположения вершин первого уровня x_1 строится расположение вершин другого уровня. Как уже было сказано, данная задача является NP-полной, и для нахождения ее приближенного решения используются эвристики, описанные ниже.

4.1. Методы, основанные на сортировке

Данная группа эвристик основана на идее сортировки вершин уровня L_2 так, чтобы уменьшить количество пересечений ребер. Легко видеть, что для двух выбранных вершин $u \in L_2$ и $v \in L_2$ количество взаимопересечений инцидентных им ребер зависит только от относительного расположения этих двух вершин на уровне L_2 и не зависит от расположения всех остальных вершин этого уровня. Таким образом, между вершинами одного уровня можно установить отношение/порядок, определяемое количеством пересечений ребер, порождаемых в различных конфигурациях расположения вершин.

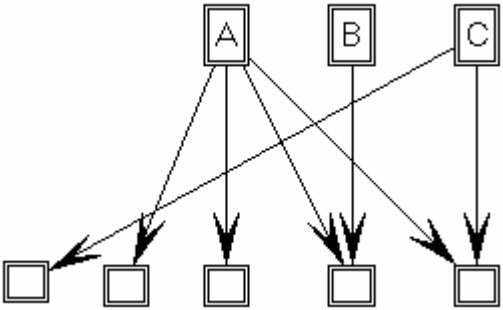


Рис. 5. Определение набора отношений между вершинами A, B и C

Понятно, что заданное таким образом отношение не может образовывать отношение линейного или частичного порядка. В противном случае обычная сортировка вершин решила бы нашу NP-полную задачу по крайней мере за квадратичное время. Иллюстрирует данное замечание следующий пример. На рис. 5 изображены два уровня иерархии с целью определения набора отношений между вершинами A, B и C. Легко видеть, что $A < B$, поскольку при расположении вершины A левее вершины B получается меньше пересечений ребер (одно против двух), $B = C$ (одно пересечение в любом из случаев), но, в свою очередь, $C < A$ (три пересечения против четырех). Получившаяся цепочка $A < B = C < A$ показывает, что заданное таким образом отношение не является транзитивным.

Однако закон трихотомии (для пары (a, b) справедливо одно и только одно из соотношений $a < b$, $a = b$, $a > b$) для нашего отношения выполняется. А это означает, что возможна устойчивая сортировка вершин, т.е. сортировка, при которой выполняются условия упорядоченности и устойчивости.

Выполнение такого упорядочивания и составляет суть методов, основанных на сортировке. Такие методы должны быстро считать потенциальное количество пересечений. Это может быть сделано напрямую за $O(|E|^2)$, а незначительными усилиями убыстрено до $O(\sum C_{uv})$, где C_{uv} — количество пересечений, образуемое парой вершин u и v. Подобный подсчет возможных пересечений для каждой пары нужно выполнить лишь один раз — это и задаст порядок на множестве вершин. Дальнейшая сортировка не изменит установившихся между вершинами отношений.

Первый из таких алгоритмов похож по своей идее на метод пузырька. Предлагается просматривать поочередно все пары соседних вершин и менять их местами, если при этом уменьшается количество пересечений. Процесс заканчивается, когда после очередного полного просмотра всех вершин второго уровня количество пересечений не уменьшилось (неупорядоченная пара не была найдена). Сложность данного алгоритма — $O(|L_2|^2)$, так как для конечного упорядочивания может потребоваться вплоть до $|L_2|$ просмотров. Результат работы такого алгоритма сильно зависит от выбора начального расположения, так как оперирует перестановками только в терминах соседних вершин.

Другой метод, позволяющий упорядочить множество вершин, основан на идее быстрой сортировки. Сначала выбирается «средняя» вершина, а все остальные вершины разбиваются на два множества в зависимости от того, в каком отношении они находятся с выбранной вершиной. Затем подобная процедура рекурсивно применяется к каждому из получившихся множеств. В худшем случае временная сложность данного алгоритма так же, как и алгоритма быстрой сортировки, является квадратичной, однако в среднем алгоритм требует всего лишь $O(|L_2| \log|L_2|)$ операций.

Идея следующего метода тоже использует концепцию сортировки вершин на уровне, хотя явно не прибегает при этом к заранее установленному отношению порядка на множестве вершин. Метод под названием «отсев» (Sifting) [62] на каждом шаге берет одну из вершин и пытается ее вставить в такое место, чтобы породить при этом наименьшее количество пересечений ребер. Для этого вершину пытаются последовательно вставить в каждое из возможных мест и считают количество получившихся пересечений. Поскольку при сдвиге вершины на одну позицию изменение количества пересечений можно посчитать за константное время (используя предварительно накопленную информацию), то алгоритм имеет квадратичную временную сложность и по своей сути похож на сортировку вставками.

Как уже было замечено, любой основанный на сортировке алгоритм упорядочивания вершин должен работать с набором отношений между па-

рами вершин, и вычисление этих отношений не может быть выполнено за линейное время. Поэтому любой из упомянутых методов будет тоже иметь нелинейную сложность.

Поскольку идеи упорядочивания заимствованы из общеизвестных методов сортировки, то авторство подобных методов чаще всего не указывается. Еще несколько методов, похожих на описанные, можно найти в [31].

4.2. Метод барицентров и метод медиан

На практике наиболее широко используются линейные алгоритмы, основанные на методе барицентров [74]. Координата вершины $v \in L_2$ определяется как среднее арифметическое координат всех ее связей с уровня L_1:

$$avg(u) = \frac{1}{\deg(u)} \sum_{v \in N(u)} x_1(v), \text{ где } N(u) := \{v \mid (u,v) \in E\}.$$

Если после перестановки координаты каких-либо двух вершин совпадают, то они разносятся на минимальное расстояние, порядок при этом определяется произвольно. Самая распространенная модификация этого метода — метод медиан [34], где вместо среднего арифметического используется координата среднего соседа с уровня x_1, т.е. если $u_1, u_2, \ldots, u_m \in L_1$ — вершины смежные v, причем $x_1(u_1) < x_1(u_2) < \ldots < x_1(u_m)$, то координата $x_2(v) = med(v)$ определяется как $x_1(u_{avg})$, где $avg = \lfloor m/2 \rfloor$. В случае, когда $med(v) = med(u)$ и v имеет нечетную степень, а u — четную, то v ставится перед u; если же четность степеней совпадает, то порядок выбирается произвольно. В работе [25] показано, что для каждой вершины u значение ее медианы может быть найдено за время $O(|N(u)|)$, что делает эту эвристику настолько же быстрой, как и метод барицентров.

Пусть $opt(G, x_1)$ — минимально возможное число пересечений ребер для данного графа, где x_1 — расположение вершин L_1, $avg(G, x_1)$ — число пересечений, остающихся после применения метода барицентров, $med(G, x_1)$ — число пересечений, остающихся после применения метода медиан. Приведем некоторые утверждения о методе барицентров и медиан (см. например [10]).

1. Если для данного графа G возможно размещение без пересечения ребер, то и метод медиан, и метод барицентров его находят, т.е. если размещение x_1 такое, что $opt(G, x_1) = 0$, тогда $avg(G, x_1) = med(G, x_1) = 0$.

2. Для любого n существует двудольный граф $G = (L_1, L_2, E)$, где $|L_1| = n$, $|L_2| = 2$, и размещение x_1 вершин L_1 такие, что $avg(G, x_1)/opt(G, x_1)$ пропорционально $n^{1/2}$.
3. Для любого n существует двудольный граф $G' = (L'_1, L'_2, E')$, где $|L'_1| = n$, $|L'_2| = 2$, и размещение x'_1 вершин L_1 такие, что $med(G', x'_1)/opt(G', x'_1) \geq 3$ — $O(1/n)$.
4. Для двудольного графа $G = (L_1, L_2, E)$ и любого размещения x_1 вершин L_1 верно неравенство $med(G', x'_1) \leq 2*opt(G', x'_1)$, если все вершины L_2 не имеют более трех ребер.

Известны также следующие модификации метода медиан.

1. *Средние медианы* [57]. Для вершин с четной степенью присваивается среднее арифметическое значение позиций вершин с соседнего уровня, и med(u) — для нечетных степеней.
2. *Полумедианы* [57]. Для вершин с четной степенью в качестве значения медианы берется значение, полученное по методу барицентров.
3. *Взвешенные медианы* [40]. Развитие метода средних медиан, отличающееся от него только в случае вершин с четной степенью, большей двух. В этом случае взвешенная медиана определяется как

$$vmed(u) = \frac{x_1(v_{j/2}) * right + x_1(v_{j/2+1}) * left}{left + right},$$

где $left = x_1(v_{j/2}) - x_1(v_1)$, и $right = x_1(v_j) - x_1(v_{j/2+1})$. Такая стратегия сдвигает вершины к той стороне, где ее соседи располагаются наиболее плотно.

4.3. Планаризация

Альтернативный подход представлен в [58]. Здесь автор предлагает заменить задачу отыскания перестановки вершин с целью минимизации пересечений ребер задачей планаризации двухуровневого ациклического графа. Под планаризацией понимается удаление минимального количества ребер, так чтобы полученный после этого двухуровневый граф можно было изобразить планарно. После нахождения необходимого минимального набора ребер граф изображается планарно, а удаленные ребра вставляются в получившуюся планарную укладку.

Понятно, что найденная таким способом перестановка вершин формально не решает задачу минимизации количества пересечений ребер, од-

нако интуитивно ясно, что полученное количество пересечений должно быть близко к минимально возможному. Причем такой алгоритм будет гарантировать достижение абсолютного оптимума в том случае, когда исходный двухуровневый граф допускает планарную двухуровневую укладку.

Аргументация такого подхода состоит в том, что визуальное восприятие запутанности графа и количества пересечений ребер зависит не только от самого числа пересечений, а также и от их взаимного расположения. Авторы подхода утверждают, что полученные таким образом представления выглядят зачастую даже лучше, чем представления, имеющие действительно минимальное количество пересечений ребер.

Несмотря на NP-полноту [35] такой постановки задачи планаризации двухуровневого графа, несомненным преимуществом перед задачей минимизации пересечений является наличие хороших эвристик ее решения с гарантированной оценкой худшего случая [75, 32].

В самой работе [58] задача планаризации двухуровневого ациклического графа формулируется в виде задачи линейного целочисленного программирования, позволяя для небольших графов находить ее точное решение.

4.4. Другие эвристики

Вероятностная эвристика [26]. Суть алгоритма заключается в «жадной вставке» вершин на основе предварительно подсчитанной для каждой вершины вероятности образования пересечений. Такая вероятность определяется суммой пересечений для каждого из ребер, инцидентных данной вершине, в полном двудольном графе.

Алгоритм Тутти [33]. Алгоритм состоит в предварительном определении позиций вершин на двух крайних уровнях с последующим решением системы разряженных линейных уравнений

$$x(u) = \frac{1}{2out\deg(u)} \sum_{v \in N^+(u)} x(v) + \frac{1}{2in\deg(u)} \sum_{w \in N^-(u)} x(w)$$

для определения координат вершин на всех промежуточных уровнях.

Назначения [21]. Данная эвристика позволяет свести задачу к известной задаче о назначениях. Для каждой вершины определяется число *dij* как верхняя оценка количества пересечений, образованных ребрами, инцидентными вершине *i*, в том случае, если вершину поместить на место j. Как и в

вероятностной эвристике *di* вычисляются путем дополнения графа до полного. Далее для матрицы $D = (dij)$ решается задача о назначениях, т.е. выбираются n элементов так, чтобы покрыть ими каждую колонку и каждый столбец матрицы и при этом минимизировать сумму их значений.

Генетический алгоритм [61]. Авторами был сконструирован генетический алгоритм для двухуровневой задачи минимизации пересечений. Проведенное сравнение с барицентрическим методом показало, что такой алгоритм дает, как правило, лучшие решения, но является крайне медленным.

4.5. Замечания по поводу практического выбора метода

Сравнению различных методов минимизации пересечений посвящено сразу несколько работ [31, 56, 49, 62]. В основном такие работы проводятся каждым из исследователей при представлении нового метода минимизации пересечений. Здесь мы постараемся просуммировать имеющиеся результаты и дать рекомендации по практическому выбору метода решения этой задачи.

Во-первых, для графов с малой насыщенностью и малым количеством вершин возможен поиск оптимального решения за приемлемое время. В этих целях используются методы целочисленного программирования. Симплекс-метод, используемый для решения этой задачи, хоть и не имеет гарантированной полиномиальной сходимости, однако на практике оказывается весьма эффективным [49].

С другой стороны, результат [10] показывает, что для насыщенных графов минимальное число пересечений ребер очень близко к максимальному. Поэтому, начиная с некоторой степени насыщенности, представляется разумным отбросить все методы поиска наилучшего расположения вершин и вместо этого взять случайное расположение.

Таким образом, простор для применения эвристик возникает лишь в «средних» случаях. Как видно из приведенных выше утверждений, метод медиан теоретически выглядит привлекательнее метода барицентров в силу наличия гарантированной верхней оценки числа пересечений. Но на практике оба метода применяются с равным успехом, при этом после применения одного из барицентрических методов разумно применение методов, основанных на сортировке для локальной корректировки и окончательной доводки расположения вершин на уровне.

Отдельно стоит задуматься над критериями остановки итерационного эвристического процесса. Чаще всего остановка производится после проведения некоторого, заранее определенного, числа итераций, либо после

того, как конфигурация вершин перестанет изменяться, либо количество пересечений перестанет уменьшаться. Для методов, основанных на сортировке, выбрать количество необходимых итераций достаточно просто, однако эта величина не может быть разумно предопределена для барицентрических методов. Изменение конфигурации вершин на каждом из шагов алгоритма очень легко отследить в процессе вычислений, но практика показывает, что останов по такому критерию наступает крайне редко. Представленные методы склонны не иметь устойчивых конфигураций вершин.

Подсчет количества пересечений на каждом шаге алгоритма кажется самым объективным критерием останова. Проблема заключается в том, чтобы быстро вычислить это количество. Подсчет в лоб может быть произведен лишь за время $O(|E|^2)$. Для методов, основанных на сортировке, такой результат вполне приемлем, поскольку одновременно с подсчетом пересечений может быть выполнено и вычисление отношения порядка на множестве вершин. В работах [15, 23] представлен алгоритм, позволяющий вычислить порядок на множестве вершин и количество фактических пересечений за время $O(E*log(E + K))$, где K — количество фактических пересечений. По сути же такой алгоритм остается квадратичным относительно числа ребер, поскольку в худшем случае количество пересечений квадратично.

Представленные квадратичные алгоритмы подсчета количества пересечений не могут быть приемлемыми для линейных барицентрических методов. Однако алгоритмы могут быть существенно убыстрены, если отказаться от одновременного вычисления отношения порядка на множестве вершин. Лучший из известных автору алгоритмов [78] производит вычисление количества пересечений ребер в двухуровневом графе за время $O(E*log(E))$, и, по всей видимости, этот результат не может быть существенно улучшен.

5. ОПРЕДЕЛЕНИЕ КООРДИНАТ ВЕРШИН НА УРОВНЕ

После определения порядка вершин на уровне необходимо определить их настоящие вертикальные координаты. Канонически, в иерархическом подходе, ребра графа изображаются в виде ломанных с точками излома, находящимися в фиктивных вершинах, поэтому задача определения окончательных координат всех вершин одновременно является и задачей проведения ребер. Если же ребра графа имеют какую-либо другую форму и/или

способ проведения, то соответствующие критерии должны быть рассмотрены при решении задачи о постановке вершин.

5.1. Точное решение

В случае изображения ребер в виде ломанных задача определения окончательных координат вершин решается из соображений минимизации числа изломов в проводимых затем ребрах. При этом должен быть учтен и не нарушен порядок следования вершин на каждом из уровней. В такой постановке задача формализуется следующим образом. Пусть у нас есть путь p = (v1, v2, … vk), в котором вершины v2, v3, …, vk-1 являются фиктивными. Если бы такой путь был изображен прямолинейным отрезком, то для i от 2 до k-1 должно выполнятся равенство углов наклона отрезков ломанной

$$y(v_i) - y(v_1) = \frac{1-i}{k-1}(y(v_k) - y(v_1)).$$

Затем, определяя функцию g(p) как

$$g(p) = \sum_{}^{k-1}(y(v_i) - a_i)^2$$
$$a_i = \frac{i-1}{k-1}(y(v_k) - y(v_1)) + y(v_1),$$

мы можем попытаться глобально минимизировать суммарное отклонение от прямолинейности: $\min \sum g(p)$, учитывая ограничение y(w) – y(z) >= q для всех пар вершин, расположенных на одном уровне так, что вершина w расположена выше вершины z. В этом случае q будет минимально возможным вертикальным расстоянием между вершинами.

Функцию для минимизации можно слегка модифицировать, например, так, чтобы отдать предпочтение ортогональным дугам. Для этого надо минимизировать квадрат разности вертикальных координат для всех ребер

$$\sum_{(u,v)\in E}(y(u) - y(v))^2 .$$

Также возможно введение дополнительных весов, которые призваны, чтобы распрямлять именно длинные ребра — как потенциальных обладателей большого количества изломов [40]

$$\sum_{(u,v)\in E}\Omega(u,v)\omega(u,v)\,|\,y(u) - y(v)\,|,$$

где $\omega(u,v)$ — мера важности спрямления ребра (u,v), $\Omega(u,v)$ — коэффициент для спрямления длинных ребер. Авторы подхода предлагают опреде-

лить $\Omega(e)$ следующим образом: $\Omega(e) = 8$, если оба конца ребра являются фиктивными вершинами, $\Omega(e) = 2$, если только одна из конечных вершин фиктивная, и $\Omega(e) = 1$ в остальных случаях.

Здесь стоит отметить два существенных отрицательных момента данного способа определения вертикальных координат вершин. Во-первых, решение данной минимизационной задачи с квадратичным относительно количества вершин набором ограничений может потребовать существенных временных ресурсов. Во-вторых, точное решение с красивыми прямолинейными ребрами может тем не менее оказаться никуда не годным ввиду чрезмерно разросшейся площади изображения. Для избежания разрастания ширины изображения можно усилить налагаемые на вертикальные координаты ограничения, однако это лишь увеличит сложность задачи и возможно элиминирует существование точного решения.

5.2. Эвристики

Ввиду перечисленных выше проблем, возникающих при попытках найти точное решение глобальной задачи спрямления ребер, на практике широко применяются эвристические подходы к решению данной проблемы.

Одной из очевидных эвристик является использование идеи барицентров в сочетании с ограничениями на порядок вершин, полученными на предыдущем шаге. Вершины последнего уровня распределяются равномерно на некотором отрезке вертикальной прямой, выделенной под вершины этого уровня. Затем для каждого следующего уровня координаты его вершин последовательно определяются как среднее арифметическое координат их соседей из уже поставленных уровней. При этом, как правило, не допускается нарушение изначального порядка вершин на уровне.

Существует также набор локальных эвристик [40], которые применяются к уже расставленным по вертикали вершинам для того, чтобы улучшить общую картину. Такие локальные эвристики, как правило, применяются последовательно до тех пор, пока не достигнет локального минимума некоторая весовая функция либо не будет совершено определенное количество итераций. В качестве оптимизируемой функции может выступать любая глобальная функция, оценивающая общую длину ребер или их общую прямолинейность.

Хорошие результаты могут быть получены применением методов, основанных на упруго-силовой физической модели [50, 37]. В таких алгоритмах положение вершины определяется силами, действующими на нее со

стороны других вершин через ребра-«пружины» или ребра-«резиновые жгуты». На такую систему накладывается ряд ограничений, для того чтобы предотвратить чрезмерное сближение вершин и сохранение предварительно вычисленного порядка вершин на уровне.

Наиболее обстоятельное и обоснованное изложение силового подхода к определению окончательных координат вершин на уровне сформулировано в [67]. Метод «маятника» интерпретирует вершины как грузы, подвешенные на ребра-струны. Далее такая система приходит в движение под действием гравитационного поля, и вершины, толкая друг друга (но не перескакивая), занимают положение, соответствующее минимуму потенциальной энергии системы. В [67] описан итерационный процесс, позволяющий смоделировать поведение такой системы и найти ее финальное положение.

Стоит отметить, что зачастую алгоритм определения вертикальных координат вершин и проведения ребер очень сильно зависит от особенностей визуализирующей системы. Так, существенным может оказаться форма изображаемых вершин, способ “прикрепления” ребер к вершинам, форма ребер и т.д. В этих случаях возможны существенные модификации алгоритма определения вертикальных координат вершин.

6. ПРЕДВАРИТЕЛЬНЫЕ ПРЕОБРАЗОВАНИЯ ГРАФА

Как уже было сказано, данная методология может быть применена не только к ациклическим графам, но также к графам, близким к ациклическим, и даже просто к неориентированным графам. В этих случаях, когда структура графа не соответствует нашему изначальному предположению об ацикличности, требуется произвести некоторые предварительные преобразования над графом. Суть и цель этих преобразований заключается в обратимом преобразовании структуры, так чтобы после размещения ациклического графа возможно было безболезненно для качества получаемого изображения восстановить структуру изначального графа и, тем самым, построить его изображение.

Существует несколько различных подходов к таким преобразованиям [20, 65, 74]. Например, можно предложить склеить каждый из циклов в одну вершину или разместить каждый из циклов на одном уровне. Возможна также вставка в цикл дополнительной вершины, которая будет его разрезать. Однако данная группа преобразований существенно изменяет структуру графа, и это, как правило, сильно сказывается на качестве полу-

чаемого изображения. Это связано с тем, что такие преобразования разрушают имеющуюся иерархическую структуру графа.

Хорошо известна проблема нахождения минимального множества дуг, разрезающих все циклы, т.е. таких дуг, выкидывание которых из графа сделает его ациклическим. Эта задача является NP-полной [38], однако для ее решения существуют хорошие линейные эвристики. К сожалению, данный метод сведения графа к ациклическому не совсем применим в случае иерархического подхода. Проблема заключается в том, что после построения изображения графа, которое не учитывает выкинутые на первом этапе ребра, задача проведения таких ребер становится слабо формализуемой и крайне тяжелой.

Следующий, общепринятый, подход к обратимому преобразованию структуры графа предлагает «разворачивать» часть ребер вместо «выкидывания» их из графа. Преимущество данного способа заключается в том, что развернутые ребра участвуют в построении финального изображения наравне с обычными родными ребрами графа. Для этих ребер также решаются задачи спрямления и минимизации пересечений. После построения конечного изображения у развернутых сначала ребер еще раз меняется ориентация. Здесь встает задача отыскания минимального множества ребер графа, разворот которых сделает его ациклическим. Стоит отметить, что, несмотря на кажущуюся существенной разницу формулировок, эта проблема эквивалентна проблеме отыскания минимального множества дуг, разрезающих все циклы. Эквивалентность предлагает использовать похожие эвристики для решения и этой задачи.

Для упрощения анализа нижеприведенных эвристик будем считать наш граф простым ориентированным графом без мультиребер. Для этого в исходном графе следует заменить мультидуги на ребра с соответсвующим весом. Отдельное внимание стоит уделить проблеме придания ориентации для 2-циклов. Если исходный граф содержит 2-циклы, т.е. пары разнонаправленных ребер $\{(u, v), (v, u)\}$, то предлагается сначала совсем удалить такие ребра, а затем к упрощенному графу применить эвристики для нахождения множества разворачиваемых ребер. После этого ребра, образующие 2-циклы, вставляются в граф и им придается одно и то же направление. В [17] показано, что такое направление всегда можно задать так, чтобы в графе не образовались новые циклы.

Наиболее простой и очевидный способ поиска дуг для разворота заключается в выделении какого-либо остовного дерева в графе и развороте тех дуг, которые не соответствуют направлению, заданному остовным деревом. В наихудшем случае при этом будет развернуто $|E| - |V| - 1$ ребер. Еще

один тривиальный метод предлагает выбрать произвольную нумерацию вершин графа и развернуть те дуги, которые идут от вершин с большими номерами к вершинам с меньшими номерами. Если более половины дуг оказывается выкинутыми, то следует поменять нумерацию на обратную, это гарантирует выкидывание не более |E| / 2 ребер в наихудшем случае.
В работе [40] предложено посчитать, сколько раз каждое ребро входит в цикл и развернуть ребро, получившее наибольшую оценку. Процесс продолжается до образования ациклического графа. К сожалению, авторы не приводят теоретических оценок для данного метода.

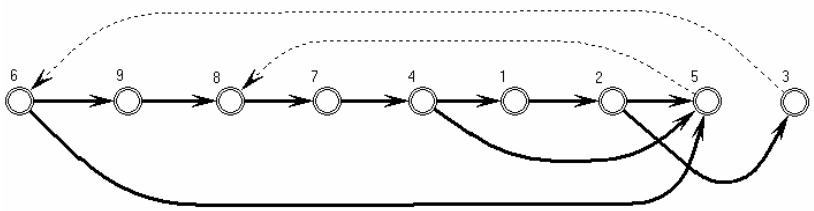


Рис. 6. Перенумерация вершин при поиске разворачиваемых ребер

Наиболее же часто используемая эвристика носит название «жадного» алгоритма [30]. Алгоритм гарантирует нахождение ациклического подграфа, в котором не менее чем (*E/2 + V/6*) ребер, и если исходный граф не имеет циклов, то алгоритм оставляет его без изменений. Эвристика предлагает найти такую нумерацию вершин, при которой количество ребер, идущих против этой нумерации, минимально (рис. 6). На каждом шаге алгоритма из исходного графа выкидывается вершина и получает свой номер. Сначала выкидываются все источники и стоки и получают, соответственно, наименьшие и наибольшие номера. В дальнейшем при отсутствии источников и стоков из графа выкидывается вершина с наибольшей разницей количества выходящих и входящих дуг и ей присваивается наименьший из свободных номеров. На каждом шаге алгоритма вместе с удалением вершины удаляются также и все инцидентные ей ребра, таким образом происходит изменение структуры графа. Существует реализация данного алгоритма, имеющая линейную временную сложность. В [29] показано, что для графов, не имеющих вершин со степенью большей 3, верхняя оценка количества разворачиваемых ребер для данного алгоритма есть (*E/3*).

В работе [69] Сандер приводит модификацию “жадного” алгоритма, основанного на другом принципе выбора следующей вершины при отсутствии источников и стоков. Вершина в данном случае выбирается сразу вме-

сте со всей своей сильно связанной компонентой (набором таких вершин u, v, что одновременно существует путь из u в v и из v в u). При надлежащем выборе структур данных такая версия алгоритма будет иметь временную сложность O(EV). Однако, несмотря на впечатляющие практические результаты, теоретическая оценка на количество разворачиваемых ребер так и не была улучшена по сравнению с оригинальной версией алгоритма.

Существуют и другие нелинейные эвристики, которые имеют лучшую оценку работы в наихудшем случае. В [29] представлено обобщение идеи Сандера. Помимо выбора сильно связных компонент, авторы используют тот факт, что для цепочек вершин достаточен разворот всего лишь одного ребра из цепочки. Стягивание цепочек вершин в исходном графе позволяет для графов, не имеющих вершин степени, большей 3, получить хорошую верхнюю оценку на количество разворачиваемых ребер — ($E/4$).

Дальнейшие поиски сложных нелинейных эвристик стоит считать малоперспективными. Во-первых, результат [64] утверждает, что вряд ли есть эвристика с хорошим наихудшим случаем. А, во-вторых, незначительное добавочное уменьшение количества развернутых ребер не оказывает существенного влияние на качество получаемого изображения.

В [14] приведена формулировка данной задачи в терминах линейного целочисленного программирования. Использовние метода ветвей и границ позволяет получить точное решение искомой задачи. Данный метод может быть применен для практического изучения результатов работы других алгоритмов.

7. ЗАКЛЮЧЕНИЕ

Отличительной чертой данной методологии является ее интуитивная понятность и масштабируемость. Под масштабируемостью мы здесь понимаем разделение методологии на ряд малопересекающихся задач. Любая финальная реализация — это всегда конструктор, т.е. набор методов, решающих задачи разных этапов. Собирая такой конструктор, исследователь должен выбрать сбалансированный набор методов. Такой набор должен быть непротиворечивым по целям и эквивалентным в терминах временной сложности. Если в набор входит алгоритм, имеющий квадратичную сложность, то нет смысла «экономить» на решении задач других этапов, применяя к ним линейные, но «плохие» эвристики.

Случаи успешного практического применения иерархического подхода к изображению ациклических графов отмечены во многих исследователь-

ских работах [5, 13, 70, 71]. Реализации данного подхода входят во многие системы визуализации графов, в том числе: GraphEd [43], D-ABDUCTOR [73], DAG [41], GD-Workbench [6], Higres [3]. Несмотря на NP-трудность почти всех возникающих подзадач, удается построить реализации, имеющие сложность около квадратичной и получающие приемлемые по качеству изображения.

Практическое сравнение различных подходов к построению размещения ациклических графов можно найти в [11]. Методы, основанные на поуровневом размещении (иерархический подход), хотя и не являются лидерами по всем эстетическим критериям, однако значительно опережают конкурентов на больших графах, возникших в реальных приложениях.

Из существенных нерешенных проблем стоит отметить задачу определения окончательных вертикальных координат для вершин. Размерность задачи линейного программирования делает использование симплекс-метода малопригодным для ее решения. Существующие же эвристики слабо формализуемы и не гарантируют отсутствия даже локальных дефектов у получающегося изображения.

Из теоретических оценок характера и свойств получаемого изображения до сих пор отсутствует какая-либо известная автору оценка площади изображения. Такая оценка до сих пор была приведена лишь для случаев планарного ациклического графа [19, 52].

СПИСОК ЛИТЕРАТУРЫ

1. **Евстигнеев В. А., Касьянов В.Н.** Базисные алгоритмы обработки бесконтурных графов. — Новосибирск, 1995. — С. 100–112.
2. **Касьянов В.Н.** Применение графов в программировании // Программирование. — 2001. — № 3. — Р.51–70.
3. **Лисицын И. А.** Применение системы HIGRES для визуальной обработки иерархических графовых моделей. // Проблемы систем информатики и программирования. — Новосибирск, 1999. — С. 64–77.
4. **Филаткина Н. Н.** Графовые средства визуализаии свойств программ. — Квалификационная работа на соискание степени бакалавра. — Новосибирск: НГУ, 1998. — 34 с.
5. **Baburin D. E., Bylyonkov M. A., Emelianov P. G., Filatkina N. N.** Visualizing Facilities in Program Reengineering. // Programming and Computer Software. — Interperiodica Publishing, 2001. — Vol. 27, N. 2. — P. 69–77.
6. **Buti L., Di Battista G., Liotta G., Tassinari E., Vargiu F., Vismara L.** GD–Workbench: A system for prototyping and testing graph drawing algorithms // Lect. Notes Comput. Sci. — 1996. — Vol. 1027. — P.111–122.

7. **Brockenauer R., Cornelsen S.** Drawing Clusters and Hierarchies // Lect. Notes Comput. Sci. — 2001. — Vol. 2025. –P. 87–120.
8. **Ball T., Eick S.** Software visualization in large // IEEE Comput. — 1996. — Vol. 29, N.4. — P. 25–39.
9. **Di Battista G., Eades P., Tamassia T., Tollis I.G.** Algorithms for drawing graphs: annotated bibliography // Comput. Geom. Theory Appl. — 1994. — P. 235–282.
10. **Di Battista G., Eades P., Tamassia T., Tollis I.G.** Algorithms for the visualization of graphs. — Prentice–Hall, 1999. — 398 p.
11. **Di Battista G., Garg A., Liotta G., Parise A., et al.** Drawing directed acyclic graphs: an experimental study // Lect. Notes Comput. Sci. — 1997. — Vol. 1190. — P.76–91.
12. **Biedl T., Kant G**. A better heuristic for ortogonal graph drawing // Lect. Notes Comput. Sci. — 1994. — Vol. 855. — P. 124–135.
13. **Di Battista G., Lillo R., Vernacotola F.** Ptolomaeus: the web cartographer // Lect. Notes Comput. Sci. — 1998. — Vol. 1547. — P. 444–445.
14. **Bastert O., Matuszewski C.** Layred drawings of digraphs // Lect. Notes Comput. Sci. — 2001. — Vol. 2025. — P. 87–120.
15. **Bentley J., Ottman T.** Algorithms for reporting and countng geometric intersections // IEEE Trans. Comput. — 1979. — Vol. 28. — P. 643–647.
16. **Bohringer K., Paulisch F. N.** Using constraints to achieve stability in automatic graph layout algorithms // Proc. of the ACM CHI'90. — 1990. — P. 43–51.
17. **Berger B., Shor P.** Approximation algorithms for the maximum acyclic subgraph problem // Proc. of the 1st ACM–SIAM Sympos. Discrete Algorithms. — 1990. — P. 236–243.
18. **Di Battista G., Tamassia R.** Algorithms for plane representations of acyclic digraphs // Theor. Comput. Sci. — 1988. — Vol. 61. — P. 175–198.
19. **Di Battista G., Tamassia R., Tollis I.G.** Area requirement and symmetry display of planar upward drawings // Discrete & Computational Geometry. — 1992. — Vol. 7. — P. 381–401.
20. **Carpano M. J.** Automatic display of hierarchized graphs for computer aided decision analysis // IEEE Trans. Syst., Man, Cybern. — 1980. — Vol.10, N. 11. — P. 705–715.
21. **Catarci T.** The assignment heuristic for crossing reduction // IEEE Trans. Syst., Man, Cybern. — 1995. — Vol. 25, N. 3. — P. 515–521.
22. **Cohen R.F., De Battista G., Tamassia R., Tollis I.G.** Dynamic graph drawings: trees, series–parallel digraphs, and planar st–graphs // SIAM J. Comput. — 1995. — Vol. 24, N. 5. — P. 970–1001.
23. **Chazelle B., Edelsbrunner H.** An Optimal Algorithm for Intersecting Line Segments in the Plane // J. ACM. — 1992. — Vol. 39, N. 1. — P. 1–54.
24. **Coffman E.G., Graham R.L.** Optimal scheduling for two processor systems // Acta Informatica. — 1972. — Vol. 1. — P. 200–213.
25. **Cormen T., Leiserson C., Rivest R.** Intorduction to algorithms // MIT Press. — Cambridge, 1990.

26. **Dresbach S.** A new heuristic layout algorithm for DAGs // Operations Research Proceedings / Ed. by Derigh, Bachem & Drexl. — Springer-Verlag. — 1994. — P. 121–126.
27. **Dobkin D.P., Gasner E.R., Koutsofios E., North S.C.** Implementing a general–purpose edge router // Lect. Notes Comput. Sci. — 1998. — Vol. 1353. — P. 262–271.
28. **Eades P., Feng Q.W., Lin X.** Straight–line drawing algorithms for hierarchical graphs and clustered graphs // Lect. Notes Comput. Sci. — 1997. — Vol. 1190. — P. 113–128.
29. **Eades P., Lin X.** A new heuristic for the feedback arc set problem // Australian J. of Combinatorics. — 1995. — Vol. 12. — P. 15–26.
30. **Eades P., Lin X., Smith W.F.** A fast and effective heuristic for the feedback arc set problem // Information Processing Letter. — 1993. — Vol. 47. — P. 319–323.
31. **Eades P., Kelly D.** Heurisitcs for reducing crossings in 2–layered networks // Ars Combin. — 1986. — Vol. 21. — P. 89–98.
32. **Eades P., McKay B., Wormald N.** On an edge crosiing problem // Proc. of the 9th Australian Comput. Sci. Conf. — 1986. — P. 327–334.
33. **Eades P., Sugiyama K.** How to draw a directed graph // J. of Information Processing. — 1990. — Vol. 13. — P. 424–437.
34. **Eades P., Wormald N.** Edge crossing in drawings of bipartite graphs // Algorithmica. — 1994. — Vol. 11. — P. 379–403.
35. **Eades P., Whitesides S.** Drawing graphs in two layers // Theor. Comput. Sci. — 1994. — Vol. 131. — P. 361–374.
36. **Frick A.** Upper bounds on the number of hidden nodes in Sugiyama's algorithm // Lect. Notes Comput. Sci. — 1997. — Vol. 1190. — P. 169–183.
37. **Fruchterman T., Reingold E.** Graph Drawing by Forcedirected Placement // Software — Practice and Experience. — 1991. — Vol. 21. — P. 1129–1164.
38. **Garey M.R., Johnson D.S.** Computers and interactability: a guide to the theory of NP–completeness / Ed. by W.H. Freeman. — New York, 1979.
39. **Garey M.R., Johnson D.S.** Crossing number is NP–complete // SIAM J. Algebraic Discrete Methods. — 1983. — Vol. 4, N. 3. — P. 312–316.
40. **Gasner E.R., Koutsofios E., North S.C., Vo K.P.** A technique for drawing directed graphs // IEEE Trans. Software Eng. — 1993. — Vol. 19, N. 3. — P. 214–230.
41. **Gasner E.R., North S.C., Vo K.P.** DAG — a program that draws directed graphs // Software — Practice and Experience. — 1988. — Vol. 18, N.1. — P. 1047–1062.
42. **Goldberg A.V., Tarjan R.E.** Finding minimum–cost circulations by successive approximation // Mathematics of Operations Research. — 1990. — Vol. 15, N. 3. — P. 430–466.
43. **Himsolt M.** GraphEd: a graphical platform for the implementation of graph algorithms // Lect. Notes Comput. Sci. — 1994. — Vol. 894. — P. 182–193.
44. **Healy P., Kuusik A.** The vertex–exchange graph: a new concept for multi–level crossing minimization // Lect. Notes Comput. Sci. — 1999. — Vol. 1731. — P. 205–216.

45. **He W., Marriott K.** Constrained graph layout // Lect. Notes Comput. Sci. — 1997. — Vol. 1190. — P. 217–233.
46. **Johnson D.** The NP–completeness column: an ongoing guide // J. on algorithms. — 1982. — Vol. 3, N. 1. — P. 215–218.
47. **Junger M., Lee E.K., Mutzel P., Odenthal T.** A polyhedral approach to the multi–layer crossing minimization problem // Lect. Notes Comput. Sci. — 1998. — Vol. 1353. — P. 13–124.
48. **Junger M., Mutzel P.** Exact and heuristic algorithms for 2–layer straightline crossing minimization // Lect. Notes Comput. Sci. — 1995. — Vol. 1027. — P. 337–348.
49. **Junger M., Mutzel P.** 2–layer straightline crossing minimization: performance of exact and heuristic algorithms // J. on Graph Algorithms and Applications. — 1997. — Vol. 1, N. 1. — P. 1–25.
50. **Kamada T., Kawai S.** An algorithm for drawing general undirected graphs // Information Processing Letters. — 1989. — Vol. 31. — P. 7–15.
51. **Kakoulis K.G., Tollis I.G.** An algorithm for labeling edges of hierarchical drawings // Lect. Notes Comput. Sci. — 1998. — Vol. 1353. — P. 169–180.
52. **Lin X., Eades P.** Area requirements for drawing hierarchically planar graphs // Lect. Notes Comput. Sci. — 1998. — Vol. 1353. — P. 219–229.
53. **Lin X.** Analysis of algorithms for drawing graphs: PhD thesis. — Dept. of Comput. Sci.; Univ. of Queensland, 1992.
54. **Lam S., Sethi R.** Worst case analysis of two scheduling algorithms // SIAM J. on Computing. — 1977. — Vol. 6, N. 3.
55. **Levialdi S.** Visual languages: where we do stand? // Lect. Notes Comput. Sci. — 1999. — Vol. 1779. — P. 145–164.
56. **Makinen E.** Experiments of drawing 2–level hierarchical graphs. — Tempere, 1988. —(Tech. Rep./ Dept. of Computer Science, University of Tampere; N A-1988-1.).
57. **Makinen E.** Experiments on drawing 2–level hierarchical graphs // Intern. J. of Comput. and Mathematics. — 1990. — Vol. 36. — P. 175–181.
58. **Mutzel P.** An alternative method to crossing minimization on hierarchical graphs // Lect. Notes Comput. Sci. — 1997. — Vol. 1190. — P. 318–333.
59. **Masuda S., Nakajima K., Kashiwabara T., Fujisawa T.** Crossing minimization in linear embeddings of graphs // IEEE Trans. on Comput. — 1990. — Vol. 39, N. 1. — P. 124 –127.
60. **Messinger E.M., Rowe L.A., Henry R.H.** A divide–and–conquer algorithm for the automatic layout of large directed graphs // IEEE Trans. on Systems, Man, and Cybernetics. — 1991. — Vol. 21, N. 1. — P. 1–12.
61. **Makinen E., Sieranta M.** Genetic algorithms for drawing bipartite graphs // Proc. of the 24th Annual ACM Symposium on the Theory of Computing, STOC'92. — 1994. — P. 527–538.
62. **Matuszewski C., Schonfeld R., Molitor P.** Using sifting for k–layer straightline crossing minimization // Lect. Notes Comput. Sci. — 1999. — Vol. 1731. — P. 205–216.

63. **North S.C.** Incremental layout in DynaDAG // Lect. Notes Comput. Sci. — 1995. — Vol. 1027. — P. 409–418.
64. **Papadimitriou C., Yannakakis M.** Optimization, approximization and complexity classes // J. of Computer and System Sciences. — 1991. — Vol. 43. — P. 425–440.
65. **Robbins G.** The ISI grapher, a portable tool for displaying graphs pictorially. — Los Angeles, 1987.— (Tech. Rep. / Information Sci. Inst., Univ. of Southern California; N ISI/RS-87-196).
66. **Reingold E., Tilford J.** Tidier drawing of trees // IEEE Trans. Soft. Eng. — 1981. — Vol. 7, N. 2. — P. 233–228.
67. **Sander G.** Graph layout through the VCG tool // Lect. Notes Comput. Sci. — 1995. — Vol. 894. — P. 194–205.
68. **Sander G.** A fast heuristic for hierarchical mangattan layout // Lect. Notes Comput. Sci. — 1995. — Vol. 1027. — P. 447– 458.
69. **Sander G.** Graph layout for applications in compiler construction. 1996 — (Tech. Rep./ Universitas des Saarlandes; N A/01/96).
70. **Seemann J.** Extending the Sugiyama algorithm for drawing UML class diagrams: towards automatic layout of object–oriented software diagrams // Lect. Notes Comput. Sci. — 1998. — Vol. 1353. — P. 415–424.
71. **Sander G., Alt M., Ferdinand C., Wilhelm R.** ClaX —a visualized compiler // Lect. Notes Comput. Sci. — 1995. — Vol. 1027. — P. 459–462.
72. **Shieh F.S., McCreary C.L.** Directed graphs drawing by clan–based decomposition // Lect. Notes Comput. Sci. — 1995. — Vol. 1027. — P. 472–482.
73. **Sugiyama K., Missue K.** A generic compound graph visualizer/manipulator: D–ABDUCTOR // Lect. Notes Comput. Sci. — 1995. — Vol. 1027. — P. 500–503.
74. **Sugiyama K., Tagawa S., Toda M.** Methods for visual undertanding of hierarchical systems // IEEE Trans. Syst., Man,and Cybern. — 1981. — Vol. 11, N. 2. — P. 109–125.
75. **Tomii N., Kambayashi Y., Shuzo Y.** On planarization algorithms of 2–level graphs // Papers of Tech. Group on Electronic computers, IECEJ. — 1977. — Vol. 38. — P. 1–12.
76. **Valls V., Marti R., Lino P.** A branch and bound algorithm for minimizing the number of crossing arcs in bipartite graphs // J. of Operational Research. — 1996. — Vol. 90. — P. 303–319.
77. **Warfield J.** Crossing theory and hierarchy mapping // IEEE Trans. Syst., Man,and Cybern. — 1977. — Vol. 7, N. 7. — P. 502–523.
78. **Waddle V., Malhotra A.** An E logE line crossing algorithm for levelled graphs // Lect. Notes Comput. Sci. — 1999. — Vol. 1731. — P. 205–216.

Т.А. Волянская

МЕТОДЫ И ТЕХНОЛОГИИ АДАПТИВНОЙ ГИПЕРМЕДИА*

ВВЕДЕНИЕ

Адаптивная гипермедиа (Adaptive Hypermedia) — относительно новое направление исследований на стыке гипермедиа и моделирования пользователя.

Одно из ограничений традиционных «статических» гипермедиа-приложений состоит в том, что они предоставляют одно и то же содержание страниц и один и тот же набор ссылок всем пользователям. Если совокупность пользователей относительно разнотипная, традиционная система будет страдать от неспособности удовлетворить потребности всех пользователей. Например, традиционная гипермедиа-система обучения предоставит одно и то же статическое объяснение и предложит одну и ту же следующую страницу студентам с широко отличающимися учебными целями и знанием темы. Статическая электронная энциклопедия предоставит одну и ту же информацию и один и тот же набор ссылок читателям с различными знаниями и интересами. Наконец, статический виртуальный музей предложит одну и ту же «экскурсию» и одни и те же комментарии посетителям с различными целями и базами знаний.

Адаптивная гипермедиа (АГ) — альтернатива традиционному подходу разработки гипермедиа-систем. Цель АГ состоит в том, чтобы увеличить функциональные возможности гипермедиа, сделав ее индивидуализированной. Адаптивные гипермедиа-системы формируют модель пользователя из знаний, целей и предпочтений индивидуального пользователя и используют ее в течение взаимодействия для адаптации к потребностям этого пользователя. Например, студенту в адаптивной гипермедиа-системе обучения будет дано представление, которое специально адаптировано к его знанию темы, и предложен набор наиболее релевантных ссылок для дальнейшего перехода. Адаптивная электронная энциклопедия индивидуализирует содержание статьи, чтобы расширить имеющиеся знания и интересы пользователя. Виртуальный музей адаптирует представление каждого про-

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

сматриваемого объекта к индивидуальному пути пользователя по музею. Это только несколько примеров использования адаптивной гипермедиа.

Данная статья содержит краткий обзор методов и технологий адаптивной гипермедиа. В первом разделе дается определение и классификация адаптивных гипермедиа-систем. Во втором разделе характеризуются прикладные области адаптивной гипермедиа, указываются их специфические особенности и определяются задачи. В третьем разделе рассматриваются различные виды адаптации, отличающиеся тем, какие аспекты принимаются во внимание системой при адаптации к пользователю. Четвертый раздел статьи посвящен краткому обзору существующих на данный момент методов и технологий адаптивного представления и адаптивной навигационной поддержки.

1. ОПРЕДЕЛЕНИЕ И КЛАССИФИКАЦИЯ АГС

Дадим следующее определение. Под *адаптивными гипермедиа-системами (АГС)* мы понимаем все гипертекстовые и гипермедиа-системы, которые отражают некоторые особенности пользователя в модели пользователя и применяют эту модель для адаптации к пользователю различных видимых аспектов системы [1,3].

Другими словами, система должна удовлетворять трем критериям:

- быть гипертекстовой или гипермедиа-системой;
- иметь модель пользователя;
- быть способной адаптировать гипермедиа, используя эту модель (т. е. одна и та же система может выглядеть по-разному для пользователей с различными моделями).

Применение адаптивной гипермедиа целесообразно в тех случаях, когда система предназначена для обслуживания большой группы пользователей с различными целями, знаниями и опытом и когда лежащее в основе гиперпространство достаточно большое. Пользователи с различными целями и знаниями могут интересоваться различными частями представленной на гипермедиа-странице информации и могут использовать различные ссылки для навигации.

Различаются следующие типы адаптации систем [5].

- *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 индивидуального пользователя.

– *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 его явному запросу. 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».
– *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 он просматривает информацию.

Большинство АГС также адаптируемы, поскольку они нуждаются в способе для инициализации модели пользователя.

2. ОБЛАСТИ ПРИМЕНЕНИЯ АГС

Анализ существующих АГС позволяет назвать шесть видов гипермедиа-систем, которые используются в настоящее время как прикладные области в большинстве научно-исследовательских проектов по адаптивной гипермедиа: гипермедиа-системы обучения, сетевые информационные системы, информационно-поисковые системы, сетевые справочные системы, институционные информационные системы и системы для управления индивидуализированными представлениями. Гипермедиа-системы обучения и сетевые информационные системы — признанные лидеры, им посвящено приблизительно две трети исследовательских работ в области АГ. Системы информационного поиска чуть менее популярны. Сетевые справочные системы и институционные гипермедиа-системы за последние несколько лет

не получили почти никакого внимания от исследователей адаптивной гипермедиа. Возможно потому, что эти виды систем все еще находятся в процессе перехода от автономной гипермедиа к Web-гипермедиа: автономные версии этих систем больше не перспективны для исследователей, а Web-версии еще недостаточно развиты [1, 3].

Гипермедиа-системы обучения (Educational Hypermedia)

Наиболее популярная область исследования для адаптивной гипермедиа — *гипермедиа-системы обучения.* Бурное развитие WWW сильно повлияло на рост исследований в этой области, движущей силой которых было обеспечение дистанционного обучения по сети. Существующие гипермедиа-системы обучения имеют относительно небольшое гиперпространство представления отдельного курса или раздела учебного материала по конкретной теме. Целью студента обычно является изучение всего материала или достаточной его части. Форма гипермедиа поддерживает управляемое студентом овладение учебным материалом. Наиболее важная характеристика пользователя в гипермедиа-системах обучения — знание пользователем преподаваемой темы.

Методы адаптивной гипермедиа могут быть полезны для решения ряда проблем, связанных с использованием гипермедиа-систем обучения. Во-первых, знания разных пользователей могут сильно различаться, и знание отдельного пользователя может весьма быстро расти. Та же самая страница может быть непонятна для начинающего и в то же время тривиальна для успевающего ученика. Во-вторых, начинающие входят в гиперпространство учебного материала, почти совсем не зная темы. Большинство предлагаемых ссылок от любого узла ведет к совершенно новому для них материалу. Они нуждаются в навигационной помощи, чтобы найти путь в гиперпространстве. Без такой помощи они могут «потеряться» даже в достаточно небольшом гиперпространстве или использовать неэффективные стратегии просмотра.

Сетевые информационные системы (On-Line Information Systems)

Другое популярное приложение для адаптивной гипермедиа — область различных *сетевых информационных систем.* Цель этих систем — обеспечить справочный доступ к информации (в отличие от систематизированного курса, как в гипермедиа-системах обучения) для пользователей с различным уровнем знания темы. Каждый узел гиперпространства обычно представляет одно понятие темы и содержит несколько страниц информации.

Подобно гипермедиа-системам обучения, сетевые информационные системы имеют проблемы с удовлетворением потребностей различных категорий пользователей. Пользователи с различными знаниями и подготовкой нуждаются в различной информации о понятии и в различных уровнях детализации. Пользователи также имеют различные цели при доступе к информационной системе. В случае, когда цель не может быть явно отображена в структуре гиперпространства или когда гиперпространство большое, пользователи нуждаются в навигационной помощи и в нахождении релевантных частей информации. Для обеспечения такой помощи система должна знать цель пользователя, определение которой — трудная задача в сетевых информационных системах, если цель не предоставляется непосредственно пользователем.

Сетевые информационные системы не формируют однородную группу систем и должны быть разделены на подгруппы. Наряду с «классическими» сетевыми информационными системами, эта группа включает ряд специализированных систем типа электронных энциклопедий, информационных киосков, виртуальных музеев, карманных руководств, систем электронной коммерции и систем поддержки производительности. Эти специализированные системы могут принимать во внимание определенный тип действий пользователя в специфической прикладной области и обеспечивать лучшую адаптивность и специальные виды адаптивного поведения.

Электронные энциклопедии (electronic encyclopedias) и *информационные киоски* (information kiosks) очень близки к определению классических сетевых информационных систем, однако они приносят больше пользы, обеспечивая некоторые специализированные расширения, которые не возможны в универсальных системах. Например, энциклопедия может отслеживать знание пользователя о различных объектах, описанных в энциклопедии, и обеспечивать адаптивные сравнения. Также она может отслеживать механизм просмотра пользователя, выводить его интересы и предлагать наиболее релевантные статьи.

Виртуальные музеи (virtual museums) и *карманные руководства* (handheld guides) сохраняют некоторое подобие с традиционными информационными системами и имеют то же самое структурированное гиперпространство объектов. Уникальная особенность этих систем — способность обеспечивать адаптивные «экскурсии» (управляемые просмотры) в этом гиперпространстве и способствовать изучению пользователем виртуального или реального музея посредством контекстно-адаптированного сопроводительного текста (комментариев). Кроме этого, карманные музейные руководства могут устанавливать положение и поведение пользователя в фи-

зическом пространстве музея. Такие карманные руководства отслеживают и поддерживают навигацию пользователя как в физическом пространстве музея, так и в виртуальном гиперпространстве. Это приводит к совершенно особому типу адаптивных гипермедиа-систем и предусматривает несколько специфических технологий моделирования пользователя и адаптации. Например, уход от объекта в середине аудиокомментария может рассматриваться как знак низкого интереса к этому объекту. Кроме того, прохождение около объекта, который мог бы быть интересен пользователю, может вызвать соответствующие комментарии.

Системы электронной коммерции (e-commerce systems) и *системы поддержки производительности* (performance support systems) весьма сильно отличаются от классических сетевых информационных систем и должны быть классифицированы как два новых вида адаптивных гипермедиа-систем. В то время как гиперпространство информационных объектов все еще составляет главную часть этих систем, просмотр гиперпространства не главное действие, а побочный продукт главного действия. Фактически, чем лучше эти системы работают, тем меньше должно требоваться просмотра. Особенно интересны адаптивные системы поддержки производительности. Эти системы можно рассматривать как комбинацию экспертных и информационных систем. Таким образом, они объединяют человеческий и искусственный интеллект при решении специфических проблем типа обеспечения лечения или технического ремонта. Поскольку эти системы обеспечивают деятельность пользователя, они имеют информацию о контексте работы пользователя и структуре целей пользователя. Это приводит к более высокому уровню точности в моделировании пользователя и высокому уровню адаптации, который раньше был возможен только в гипермедиа-системах обучения и сетевых справочных системах.

Сетевые справочные системы (On-Line Help Systems)

Сетевые справочные системы очень близки к сетевым информационным системам. Эти системы предоставляют интерактивную информацию о компьютерных прикладных программах, необходимую для помощи пользователям этой системы. Отличие от прежней категории состоит в том, что сетевые справочные системы не автономны, как сетевые информационные системы, а связаны со своей прикладной системой. Другое отличие в том, что гиперпространство в существующих сетевых справочных системах относительно небольшое.

Сетевые справочные и информационные системы разделяют проблему предоставления различной информации различным пользователям. В то же время проблема помощи пользователям в нахождении релевантных частей информации менее важна для сетевых справочных систем, поскольку гиперпространство небольшое и поскольку система знает контекст, в котором пользователь запросил справку (контекстно-зависимая справка). Контекст работы в прикладной системе обеспечивает надежный источник информации для адаптивной сетевой справочной системы, чтобы определить цель пользователя и предложить наиболее релевантные пункты справки.

Информационно-поисковые гипермедиа-системы (IR hypermedia)

Информационно-поисковые (ИП) гипермедиа-системы — класс ИП систем, которые объединяют традиционные технологии информационного поиска с доступом, подобным гипертексту, от индексных элементов к документам и обеспечивают возможность просмотра гиперпространства документов, используя отношения подобия между документами. Известно, что просмотр может помогать пользователям находить требуемые документы, когда они имеют проблемы с построением правильного формального запроса. Размер гиперпространства в регулярной ИП гипермедиа обычно очень большой и не может быть структурирован «вручную». Это означает, что ссылки в этом гиперпространстве не поставляются разработчиками, как в сетевых информационных системах, а рассчитываются системой, например, при использовании измерения отношения подобия (между двумя документами предоставляется ссылка, если оба документа удовлетворяют некоторому условию подобия). Другое отличие от сетевых информационных систем состоит в том, что пользователи ИП гипермедиа наиболее часто являются профессионалами в различных областях и используют систему в своей повседневной работе с различными ИП целями. Адаптивные ИП гипермедиа-системы могут предложить некоторую дополнительную помощь, ограничивая навигационный выбор и предлагая следовать по наиболее релевантным ссылкам.

Рост WWW значительно повлиял на развитие гипермедиа-систем информационного поиска. Наиболее перспективная задача в этом классе систем состоит в том, чтобы поддержать процесс информационного поиска пользователя в неограниченном гиперпространстве WWW. В то время как несколько интересных систем были разработаны для «классической» замкнутой совокупности параметров (в контексте WWW это обычно означает одиночный Web-сайт), большая часть систем пытается обрабатывать «весь»

WWW. Среди большого количества адаптивных ИП гипермедиа-систем, разработанных до настоящего времени, можно выделить две большие группы — системы, *ориентируемые на поиск* (search-oriented systems), и системы, *ориентируемые на просмотр* (browsing-oriented systems), и несколько подгрупп в пределах каждой группы.

Цель *ориентируемых на поиск* систем состоит в формировании списка ссылок к документам, удовлетворяющим текущему информационному запросу пользователя. В отличие от простых «одноразовых» поисковых машин, адаптивные ИП системы принимают во внимание не только набор слов, определяющих текущий запрос, но также и долгосрочную (или краткосрочную) модель интересов и предпочтений пользователей. Различается два вида ориентируемых на поиск систем: к*лассические ИП системы,* имеющие дело с замкнутой совокупностью информационного пространства, и *поисковые фильтры* (search filters), работающие с неограниченным WWW. Они расширяют возможности существующих «одноразовых» Web-машин поиска, применяя различные основанные на модели подходы адаптивной навигационной поддержки (типа удаления и аннотирования ссылок) к результатам поиска, чтобы помочь пользователю выбрать наиболее релевантные ссылки для дальнейшего просмотра.

Системы, *ориентируемые на просмотр,* поддерживают пользователей в процессе просмотра, управляемого поиском. Как и в других типах адаптивных гипермедиа-систем, это реализуется с помощью стандартных технологий адаптивной навигационной поддержки. *Системы адаптивного руководства* (adaptive guidance systems) отмечают одну или несколько ссылок на текущей странице, которые являются наиболее релевантными целями пользователя. *Системы адаптивного аннотирования* (adaptive annotation systems) добавляют различные визуальные подсказки к ссылкам на текущей странице, чтобы помочь пользователю выбрать наиболее релевантную ссылку. *Системы адаптивных рекомендаций* (adaptive recommendation systems) пытаются вывести цели и интересы пользователя из его стратегии просмотра и сформировать список рекомендуемых ссылок к узлам, которые не могут непосредственно быть достигнуты с текущей страницы, но наиболее подходят пользователю.

Между адаптивными системами рекомендаций, работающими в информационном пространстве замкнутой совокупности и работающими с неограниченным WWW, существует важное различие. Системы рекомендаций, работающие в информационном пространстве замкнутой совокупности, могут формировать исчерпывающий список из ссылок к наиболее релевантным узлам. Системы рекомендаций, работающие в гиперпростран-

стве открытой совокупности, предлагают релевантные ссылки, превращая некоторую значимую область полного гиперпространства в гиперпространство закрытой совокупности и изучая структуру и содержание узлов в этой области. В настоящее время можно выделить два способа «закрытия» гиперпространства открытой совокупности. В однопользовательских системах небольшая часть WWW обычно анализируется на несколько шагов вперед от текущей точки просмотра пользователя, впоследствии система может рекомендовать пользователю наиболее релевантные ссылки из выбранной области. В многопользовательских системах анализируется совокупность данных просмотра группы пользователей.

Возможности использования одних и тех же механизмов адаптации для обеспечения различных видов поддержки пользователя в ИП контексте могут демонстрироваться *информационными Web-службами* (Web-based information services), новым классом ИП гипермедиа-систем [1]. Информационные службы работают, собирая общий банк документов (URL) из гиперпространства открытой совокупности в течение длительного периода времени. В отличие от ранних прототипов информационных служб, современные службы работают с группой пользователей и имеют возможность изучить и банк данных пользователей, и банк данных документов. Информационные службы обычно формируются с использованием технологии агентов. Поддерживающие пользователя агенты наблюдают за действиями пользователей и обеспечивают адаптивное предложение в соответствии с краткосрочной информационной потребностью и долгосрочной моделью интересов. Набор ссылок может быть собран двумя различными путями. *Службы фильтрации* (filtering services) работают с существующим потоком входящих документов типа статей новостей или финансовых объявлений. *Службы поиска* (search services) используют искусственных или реальных агентов, знающих интересы пользователя, для выполнения активного поиска новых документов в WWW. Информационные службы имеют потенциал для предоставления всех известных типов ИП гипермедиа-сервисов от поиска до управления индивидуализированными представлениями. Кроме того, информационная служба может выполнять функции, подобные универсальной сетевой информационной системе в пределах банка замкнутой совокупности документов.

Системы для управления индивидуализированными представлениями в информационных пространствах (Systems for Managing Personalized Views in Information Spaces)

Существующие телекоммуникационные системы типа WWW предлагают огромное количество различных информационных и интерактивных услуг, которые формируют действительно неограниченные гиперпространства. Многие пользователи должны иметь доступ к одному или нескольким подмножествам всего гиперпространства для своей повседневной работы. Чтобы защитить себя от сложности полного гиперпространства, они заинтересованы в определении индивидуализированных представлений относительно всего гиперпространства. Каждое представление может быть посвящено одной из целей или интересов, связанных с работой пользователя. Частично эта прикладная область подобна институционной гипермедиа и другим видам информационных систем, в которых пользователи нуждаются в удобном доступе к подмножеству информационного пространства для повседневной работы. Новый фактор, который вынуждает системы иметь дело с широкими (мировыми) информационными пространствами, это динамический характер гиперпространства, в котором элементы могут появляться, исчезать или изменяться. Индивидуализированные представления в мировых информационных пространствах требуют постоянного управления, т.е. поиск новых и релевантных элементов и распознавание измененных элементов или элементов с истекшим сроком хранения (этим данная прикладная область подобна ИП гипермедиа).

Можно назвать, по крайней мере, два стандартных механизма для управления индивидуализированными представлениями*: индивидуализированные представления сайтов* (personalized site views) (типа MyYahoo или MyNetscape) и *организаторы закладок* (bookmark organizers). Однако большая часть индивидуализированных сайтов и организаторов закладок адаптируема, но не адаптивна.

Институционные информационные системы (Institutional Hypermedia)

Другая область приложения для адаптивной гипермедиа — институционные информационные системы, которые интерактивно предоставляют всю информацию, требуемую для поддержания работы некоторого учреждения. Первоначально эти виды систем были разработаны как набор свободно связанных баз данных, но в некоторых недавних системах такие базы данных соединены в единое гиперпространство, которое может быть достаточно большим. Специфическая особенность этих систем в том, что они

являются средой для повседневной работы многих служащих учреждения. Согласно своей профессии, они могут всегда использовать только определенную область гиперпространства, и, согласно текущей рабочей цели, они могут нуждаться в доступе к очень небольшому его подмножеству. Большинству пользователей никогда не потребуется обращаться к частям гиперпространства вне своей рабочей области, кроме того, слишком большое количество навигационных возможностей отвлекает их от основной работы. В этом отношении работоориентированные институционные информационные системы значительно отличаются от ИП гипермедиа и сетевых информационных систем, где «рабочая область» пользователя — все гиперпространство. В то же время пользователи институционных информационных систем могут нуждаться в помощи по организации более удобного индивидуализированного доступа к своим рабочим областям.

Другая проблема институционных информационных систем, которая схожа с одной из проблем гипермедиа-систем обучения, связана с новыми служащими, не знакомыми со структурой гиперпространства (хотя они могут быть знакомы с прикладной областью непосредственно), и которые могут «потеряться» даже в небольшой профессиональной подобласти.

3. ВИДЫ АДАПТАЦИИ

Следующий вопрос, который следует задать при разговоре о конкретном виде адаптивной системы: «Какие аспекты могут быть приняты во внимание при обеспечении адаптации?» К каким особенностям, которые могут быть различны для разных пользователей (и могут быть различны для того же самого пользователя в определенное время), система может адаптироваться?

Традиционно адаптация в адаптивных системах была основана на принятии во внимание различных характеристик пользователей, представленных в модели пользователя [1, 3]. В настоящее время ситуация другая: ряд адаптивных Web-систем способен адаптироваться не только к характеристикам пользователя. Предлагается различать адаптацию к данным пользователя (user data), рабочим характеристикам (usage data) и данным окружения (environment data) [1]. *Данные пользователя* включают различные характеристики пользователей. *Рабочие характеристики* включают данные о взаимодействии пользователя с системами, которые не могут быть сведены к характеристикам пользователя (но все еще могут использоваться для принятия решений адаптации). *Данные окружения* включают все аспекты поль-

зовательского окружения, которые не связаны с пользователями непосредственно.

Адаптация к данным пользователя

Имеется много особенностей, связанных с текущим контекстом работы пользователя и с пользователем как индивидуумом, которые могут быть приняты во внимание адаптивной системой. Наиболее часто адаптивными гипермедиа-системами используются следующие характеристики пользователя: знания, цели, подготовка, опыт в гиперпространстве, предпочтения, интересы и индивидуальные особенности пользователя.

Знания

Знание пользователем темы, представленной в гиперпространстве, по-видимому, наиболее важная характеристика пользователя для существующих адаптивных гипермедиа-систем. Почти все технологии адаптивного представления полагаются на знание пользователя как источник адаптации. Поскольку знание пользователя непостоянно и изменчиво для отдельного пользователя, адаптивная гипермедиа-система должна распознавать изменения в состоянии знания пользователя и соответственно обновлять модель.

Знание пользователем темы наиболее часто представляется *оверлейной моделью* (overlay model), которая основана на структурной модели предметной области [5]. Структурная модель (structural model) представляется как множество связанных между собой понятий, формирующих своего рода семантическую сеть, которая представляет структуру предметной области. Понятия могут называться по-разному в различных системах — темами, разделами, элементами знания, объектами, результатами обучения, но во всех случаях они являются элементарными частями знания для данной предметной области. Смысл *оверлейной модели* состоит в том, чтобы представить знание темы индивидуальным пользователем как перекрытие («оверлей») модели предметной области. Для каждого понятия модели области оверлейная модель индивидуума хранит некоторое значение, которое является оценкой уровня знаний пользователем этого понятия. Это может быть двоичное значение (известное — неизвестное), качественная мера (хорошее — среднее — плохое) или количественная мера, типа вероятности того, что пользователь знает понятие. Оверлейная модель знаний пользователя может быть представлена как набор пар «понятие — значение», по одной паре для каждого понятия области. Оверлейные модели мощны и

гибки, они могут независимо измерять знание пользователем различных тем.

Иногда используется более простая *стереотипная модель* (stereotype model) пользователя для представления его знаний [1, 3]. Стереотипная модель различает несколько типовых, или «стереотипных» пользователей. Для каждого аспекта моделирования пользователя система может иметь набор возможных стереотипов (шаблонов). Конкретный пользователь обычно моделируется с помощью причисления к одному из стереотипов для каждого аспекта моделирования. Стереотипная модель пользователя также может быть представлена как набор пар «стереотип—значение», где значение может быть не только «истиной» или «ложью» (что означает, что пользователь принадлежит или не принадлежит стереотипу), но и равняться некоторому вероятностному значению (которое представляет вероятность того, что пользователь принадлежит стереотипу). Стереотипная модель более простая и менее мощная, чем оверлейная модель, но она также более общая и намного проще для инициализации и обслуживания.

Проблема со стереотипной моделью знаний состоит в том, что многие эффективные технологии адаптации требуют более мелкомодульной оверлейной модели. В свою очередь, оверлейная модель пользователя имеет проблему инициализации, поскольку очень трудно установить все значения после короткого интервью с новым пользователем. Хорошие результаты могут быть достигнуты путем комбинации стереотипного и оверлейного моделирования. Они могут быть объединены следующим способом: в начале работы для классифицирования нового пользователя и установления начальных значений для оверлейной модели используется стереотипное моделирование, затем используется обычная оверлейная модель.

Цели

Цель пользователя или задача пользователя — характеристика, связанная с контекстом работы пользователя в гипермедиа, скорее, чем с пользователем как индивидуумом. В зависимости от вида системы, это может быть цель работы (в прикладных системах), цель поиска (в информационно-поисковых системах), решение задач или цель обучения (в гипермедиа-системах обучения). Цель пользователя — наиболее изменчивая характеристика пользователя: почти всегда она изменяется от сеанса к сеансу и часто может изменяться несколько раз в пределах одного сеанса работы. В некоторых системах разумно различать локальные цели, или цели нижнего уровня, которые могут изменяться весьма часто, и общие цели, или цели высшего уровня, которые являются более устойчивыми. Например, в ги-

пермедиа-системах обучения цель обучения — общая цель, в то время как решение задач — локальная цель, которая изменяется от одной учебной задачи до другой несколько раз в пределах сеанса.

Текущая цель пользователя обычно моделируется способом, несколько подобным оверлейному моделированию знаний. Как правило, каждая система поддерживает набор возможных целей или задач пользователя, которые она может распознавать. Более развитые «цель-основанные» системы используют расширенное представление возможных и текущих целей пользователя. Наиболее расширенное представление возможных целей пользователя — иерархия (дерево) задач. Наиболее расширенное представление текущих целей пользователя — набор пар «цель—значение», где значение — вероятность того, что соответствующая цель — текущая цель пользователя.

Подготовка и опыт

Две особенности пользователя, которые сходны со знанием пользователем темы, но функционально отличаются от него, — подготовка и опыт пользователя в данном гиперпространстве. Под *подготовкой пользователя* мы понимаем всю информацию, связанную с предыдущим опытом пользователя вне темы гипермедиа-системы, достаточно уместную для рассмотрения. Например, профессию, опыт работы в связанных областях, а также точку зрения и перспективы пользователя.

Под *опытом пользователя* в данном гиперпространстве мы понимаем знание пользователем структуры гиперпространства и его навигационные возможности. Это не то же самое, что знание пользователем темы. Иногда пользователь, который хорошо знаком непосредственно с темой, вообще не знаком со структурой гиперпространства. Наоборот, пользователь может быть хорошо знаком со структурой гиперпространства без глубокого знания темы. Еще одна причина отличать опыт в гиперпространстве от уровня знаний — существование технологии адаптивной навигации, которая полагается на эту особенность пользователя. Подготовка и опыт обычно также моделируются стереотипной моделью пользователя.

Предпочтения

По различным причинам пользователь может предпочитать некоторые узлы, ссылки и части страницы другим. Эти предпочтения могут быть абсолютными или относительными, т. е. зависящими от текущего узла, цели и текущего контекста вообще. Наиболее интенсивно предпочтения исполь-

зуются в информационно-поисковых гипермедиа-системах, в которых предпочтения — часто единственная хранимая о пользователе информация.

Предпочтения пользователя отличаются от других компонентов модели пользователя по нескольким аспектам. В отличие от других компонентов, предпочтения не могут быть выведены системой. Пользователь должен сообщить системе непосредственно или косвенно (простой обратной связью) о своих предпочтениях. Это больше похоже на адаптируемость, чем на адаптивность. Отличие в том, что адаптивные гипермедиа-системы могут выводить предпочтения пользователя и применять их для адаптации в новых контекстах.

Другая специфическая особенность моделирования предпочтений — способ представления. В то время как другие части модели пользователя обычно представляются символически, предпочтения часто представляются численно и рассчитываются специальными способами. Цифровой способ представления имеет некоторые преимущества перед символическим способом: он открывает возможность объединения нескольких моделей пользователей и их суммирования в групповую модель пользователей. Групповые модели накапливают предпочтения определенной группы и таким образом являются хорошей стартовой моделью для нового члена группы.

Интересы

Интересы пользователя — характеристика, не используемая в ранних адаптивных гипермедиа-системах. Ситуация резко изменилась с ростом информационно-поисковых Web-систем, которые пытаются смоделировать долгосрочные интересы пользователя и использовать их параллельно с краткосрочной целью поиска для улучшения фильтрации информации и рекомендаций. Также использование этой характеристики становится популярным в различных сетевых информационных системах типа информационных киосков, электронных энциклопедий и музейных гидов. В этих системах интересы пользователя служат основанием для рекомендации релевантных гиперузлов.

Индивидуальные особенности

Индивидуальные особенности пользователя — характеристики пользователя, которые определяют пользователя как индивидуума, например, индивидуальные показатели (интроверт — экстраверт), познавательные факторы и стиль обучения. Подобно подготовке пользователя, индивидуальные особенности — стабильная характеристика пользователя, которая не изме-

няется вообще или изменяется в течение длительного периода времени. В отличие от подготовки пользователя, индивидуальные особенности традиционно извлекаются не простым интервью, а специально разработанными психологическими тестами. Большинство исследователей соглашается с необходимостью моделирования и использования индивидуальных особенностей, но разногласия возникают насчет того, какие особенности могут и должны использоваться и как их использовать. Кроме того, несколько экспериментальных исследований, проводимых с целью выяснить, есть ли смысл обрабатывать пользователей с различными индивидуальными особенностями по-разному, не выявили никаких существенных различий.

Адаптация к данным окружения

Адаптация к окружению пользователя — новый вид адаптации, который был привнесен Web-системами [1]. Поскольку пользователи одного и того же серверного Web-приложения виртуально могут постоянно находиться всюду, а также использовать различное оборудование, адаптация к окружению пользователя стала важной задачей. Ряд современных адаптивных гипермедиа-систем предложил некоторые технологии для адаптации как к местонахождению пользователя, так и к платформе пользователя. Простая адаптация к платформе (аппаратные средства, программное обеспечение, пропускная способность сети) обычно включает выбор типа данных и средств (т. е. графика против видео) для представления содержания. Более прогрессивные технологии могут обеспечивать различный интерфейс пользователям с разнообразными платформами. Адаптация к местонахождению пользователя может успешно использоваться многими сетевыми информационными системами. Наиболее интересный тип приложений для исследования этой стороны адаптации — карманные руководства. Современные исследования в этой области предложили ряд интересных технологий адаптации, принимающих во внимание местонахождение, направление взгляда и движения пользователя.

4. ТЕХНОЛОГИИ АДАПТАЦИИ

В адаптивной гипермедиа область адаптации весьма ограничена: не так много характеристик, которые могут быть изменены. На некотором уровне обобщения, гипермедиа состоит из набора узлов или гипердокументов («страниц»), связанных ссылками. Каждая страница содержит некоторую локальную информацию и ряд ссылок к связанным страницам. Гипермедиа-

системы могут также включать индекс и глобальную карту, которые обеспечивают ссылки ко всем доступным страницам. Все, что может быть адаптировано в адаптивной гипермедиа, — это содержание обычных страниц (*адаптация на уровне содержания*) и ссылки с обычных страниц, страниц индексов и карт (*адаптация на уровне ссылок*). Адаптации на уровне содержания и ссылок дифференцируются как два различных класса адаптации гипермедиа. Первый называется *адаптивным представлением* (adaptive presentation), а второй — *адаптивной навигационной поддержкой* (adaptive navigation support) [1, 3].

Адаптивное представление

Смысл различных технологий адаптивного представления состоит в том, чтобы адаптировать содержание страницы, к которой обращается отдельный пользователь, к текущему знанию, целям и другим характеристикам пользователя. Например, квалифицированному пользователю можно предоставить более детальную и глубокую информацию, в то время как начинающий может получать дополнительные объяснения.

Адаптивное представление — общий термин для всех технологий, которые адаптируют содержание гипермедиа-страницы в соответствии с моделью пользователя. Различаются следующие представления ([1–3, 5]):

- *адаптивное представление текста* — текстовое содержание гипермедиа-страницы изменяется в зависимости от модели пользователя, т. е. пользователи с различными моделями получают различные тексты в качестве содержания одной и той же страницы;
- *адаптивное представление мультимедиа* — хотя этот термин предполагает, что элементы мультимедиа-содержания могут быть адаптированы к индивидуальному пользователю, текущие реализации ограничены выбором средств. В отличие от текста, содержание анимации, аудио или видео фрагментов не адаптировано;
- *адаптация модальности* — новая технология адаптации содержания высокого уровня [1]. 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 обра-

зом 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.

Методы адаптивного представления

Содержание документа адаптируется к потребностям пользователя посредством сокрытия некоторой специализированной информации или добавления дополнительной информации. Рассмотрим далее основные методы адаптивного представления текста: дополнительные, предварительные и сравнительные объяснения, варианты объяснения и сортировку [1–5].

Дополнительные объяснения (Additional Explanations)

Цель метода *дополнительных объяснений,* наиболее популярного метода адаптации содержания, показать пользователю те части информации об отдельном понятии, которые соответствуют его знаниям или целям, и скрыть те, которые не соответствуют.

Например, подробности нижнего уровня могут быть скрыты от пользователей с плохим уровнем знания этого понятия, поскольку они не смогут их понять. Напротив, дополнительные пояснения, обычно требующиеся начинающим для понимания понятия, могут быть скрыты от пользователей с хорошим уровнем знания понятия, поскольку они больше в них не нуждаются. В общих чертах, вдобавок к основному представлению, некоторая категория пользователей может получить дополнительную информацию, которая специально подготовлена для этой категории и не будет показываться пользователям других категорий.

Предварительные объяснения (Prequisite Explanations)
и сравнительные объяснения (Comparative Explanations)

Два других метода — *предварительные (необходимые как условия)* и *сравнительные объяснения* — изменяют представляемую о понятии информацию в зависимости от уровня знаний пользователем сходных (близких) понятий.

Первый метод основан на предварительных связях между понятиями. Идея в следующем: перед предоставлением объяснения понятия система

добавляет объяснения всех понятий, предварительно необходимых для понимания этого понятия, которые еще не известны пользователю.

Второй метод основан на отношении подобия (сходства) между понятиями. Смысл метода состоит в том, чтобы при объяснении нового понятия подчеркивать его связь с уже известными понятиями. Если понятие, сходное с представляемым понятием, известно, пользователь получает сравнительное объяснение, которое подчеркивает сходства и различия между текущим и сходным понятиями.

Варианты объяснения (Explanation Variants)

Очевидно, что для адаптации не всегда достаточно показа или сокрытия некоторых частей информации, поскольку различные пользователи могут нуждаться в существенно различной информации. При использовании метода вариантов объяснения система хранит несколько различных вариантов объяснения отдельного понятия, и пользователь получает вариант, наиболее соответствующий своей модели.

Сортировка (Sorting)

Интересный метод, который может принимать во внимание как подготовку, так и уровень знаний пользователя — *сортировка* фрагментов информации о понятии. Фрагменты информации сортируются и отображаются на странице от наиболее до наименее релевантного подготовке и знанию пользователя. Метод наиболее эффективен для информационно-поисковых систем.

Технологии адаптивного представления

Следующие технологии используются для реализации перечисленных выше методов адаптивного представления текста [1–5].

Условный текст (Conditional Text)

Простая, но эффективная технология для адаптации содержания — технология *условного текста*. При использовании этой технологии вся возможная информация о понятии разделена на несколько частей текста. Каждая часть связана с условием на уровне знания пользователя, представленного в модели пользователя. При представлении информации о понятии система показывает только те части текста, для которых условие истинно. Это технология нижнего уровня, требующая некоторой работы «програм-

мирования» от автора для установления всех требуемых условий. Выбирая соответствующие условия на уровне знаний текущего понятия и связанных понятий, представленных в модели пользователя, можно реализовать все методы адаптации, перечисленные выше, за исключением сортировки. Простой пример — это сокрытие 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.

Расширяющийся текст (Stretchtext)

Технология более высокого уровня, которая может также включать и выключать различные части текста в соответствии с уровнем знания пользователей, основана на расширяющемся тексте (stretchtext), который является специальным видом гипертекста. В обычном гипертексте результатом активации ключевого слова является перемещение на другую страницу со связанным текстом. В stretchtext 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 — stretchtext-страница, которая может содержать много «свернутых» слов. Идея адаптивного stretchtext-представления состоит в том, чтобы представить требуемую страницу так, чтобы все stretchtext-расширения нерелевантные для пользователя были свернуты, а все расширения релевантные для пользователя, — развернуты.

Важная особенность адаптивной stretchtext-технологии в том, что она позволяет и пользователю, и системе адаптировать содержание отдельной страницы и что она может принимать во внимание и знания, и предпочтения пользователя. После произвольного представления stretchtext-страницы она может быть далее адаптирована пользователем, который имеет возможность 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.

Варианты фрагмента (Fragment Variants)
и варианты страницы (Page Variants)

Метод вариантов объяснения может быть реализован с помощью технологий вариантов фрагмента и вариантов страницы.

Варианты фрагмента — более мелкомодульная реализация метода *вариантов 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 — 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.

Фреймовая технология (Frame-Based Technique)

Наиболее мощная из всех технологий адаптации содержания — *фреймовая* (основанная на фреймовом представлении) *технология*. 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 используются для выбора одной из существующих схем представления (каждая схема — упорядоченное подмножество слотов), и затем эта схема 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.

Фреймовая технология 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.

Адаптивная навигационная поддержка

Смысл технологий адаптивной навигационной поддержки состоит в том, чтобы помочь пользователям найти путь в гиперпространстве с помощью адаптации способа представления ссылок к целям, знанию и другим характеристикам индивидуального пользователя. Эти технологии 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) [1–5].

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 может быть изменено и адаптировано [3].

1. *Локальные неконтекстные (контекстно-независимые) ссылки.* Этот тип включает все виды ссылок на обычных гипермедиастраницах, 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.

Методы адаптивной навигационной поддержки

Технологии адаптивной навигационной поддержки используются для достижения нескольких целей адаптации: обеспечить глобальное руководство, локальное руководство, локальную ориентацию, глобальную ориентацию и управление индивидуализированными представлениями в информационных пространствах [1,3].

Глобальное руководство (Global Guidance)

Глобальное руководство 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, кото-

рая обычно полностью или частично предоставляется им, — 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) для того, чтобы сделать свободный выбор.

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.

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 АГ системы осуществляют поддержку локальной ориентации двумя различными способами: предоставляя дополнительную информацию об узлах, доступных от текущего узла (использование технологии *аннотирования*), и ограничивая число навигационных воз-

можностей для уменьшения познавательной перегрузки, при этом пользователям позволено 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.

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

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.

Технологии адаптивной навигационной поддержки

Полное руководство (Direct Guidance)

Полное руководство — 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:

- «следующий наилучший» (*next best*) — предоставление кнопки «next» для навигации через гиперпространство,
- установление последовательности страниц или след (*page sequencing or trails*) — 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.

Адаптивная сортировка (упорядочение) ссылок (Adaptive Sorting (Ordering) Of Links)

Вместо предоставления единственной «наилучшей» ссылки, эта технология предоставляет список ссылок, упорядоченный по убыванию релевантности, в соответствии с моделью пользователя. Различаются два метода сортировки:

- сортировка по подобию (*similarity sorting*) — ссылки сортируются, исходя из их подобия данной странице;
- предварительные знания (*prerequisite knowledge*) — 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.

Адаптивное сокрытие ссылок (Adaptive Link Hiding)

Адаптивное сокрытие ссылок — 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.

Различаются три вида сокрытия ссылок: непосредственно *сокрытие ссылок* (ссылка делается неотличимой от нормального текста), *удаление ссылок* (ссылка делается невидимой) и *отключение ссылок* (ссылка делается недоступной) [1, 5].

Адаптивное сокрытие ссылок (adaptive link hiding) — эта технология 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 (adaptive link removal) — эта технология удаляет анкер ссылки для нерелевантных ссылок. Это может быть выполнено только тогда, когда окружающий текст все еще имеет смысл после удаления ссылки. Хотя адаптивное удаление ссылок эффективно сокращает число навигационных шагов, предварительные оценки показывают, что пользователям не нравится эта технология.

Адаптивное отключение ссылок (adaptive link disabling) — функциональные возможности ссылки удаляются, но текст ссылки остается. Эта технология 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.

Адаптивное аннотирование ссылок (Adaptive Link Annotation)

Смысл технологии *адаптивного аннотирования ссылок* 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 недос-

таточно 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 несколько 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 — уместное и неуместное, 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 улучшения понимания учебного материала.

Адаптивное генерирование ссылок (Adaptive Link Generation)

Рост систем рекомендаций делает необходимыми два 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 предлагается рассматривать как новую технологию адаптивной навигационной поддержки высокого уровня [1]. Эта технология может использоваться вместе с существующими технологиями типа аннотирования и сортировки.

Адаптация карты (Map Adaptation)

Технология *адаптации карты* включает различные пути адаптации формы глобальных и локальных гипермедиа карт (графические представления структуры ссылок), представленных 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 адаптирования гипермедиа-карты, но все эти технологии не изменяют форму или структуру карт.

Полное руководство, сортировка, сокрытие, аннотирование и адаптация карты — 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 из трех других технологий.

ЗАКЛЮЧЕНИЕ

В статье были рассмотрены системы, методы и технологии адаптивной гипермедиа относительно нового направления исследований в области гипермедиа и моделирования пользователя, альтернативного подхода к разработке гипермедиа-приложений.

В отличие от традиционных гипермедиа-систем, предлагающих одно и то же содержание и одну и ту же структуру навигации всем пользователям, адаптивные гипермедиа-системы дают возможность предоставления «индивидуализированных» представлений гипермедиа. Адаптивные гипермедиа-системы — это гипермедиа-системы, которые отражают некоторые особенности пользователя в модели пользователя и применяют эту модель для адаптации к пользователю различных видимых аспектов системы.

Методы адаптивной гипермедиа применяются во многих прикладных областях: гипермедиа-системах обучения, сетевых информационных системах, информационно-поисковых системах, справочных системах и т.д.

АГС могут адаптироваться к различным характеристикам индивидуального пользователя: знанию, целям, опыту, предпочтениям, интересам, окружению и т.д. Различаются два класса адаптации гипермедиа: адаптивное

представление — адаптация текстового и мультимедиа содержания гипермедиа-страниц и адаптивная навигационная поддержка — адаптация ссылок и структуры навигации. На данный момент существует ряд эффективных методов и технологий для адаптивного представления и адаптивной навигационной поддержки, краткий обзор которых представлен в статье.

СПИСОК ЛИТЕРАТУРЫ

1. **Brusilovsky P.** Adaptive Hypermedia // User Modeling and User-Adapted Interaction. — 2001. — Vol. 11. — P. 87–110.
2. **Brusilovsky P.** Methods and techniques of adaptive hypermedia. // Adaptive Hypermedia and Hypermedia. — Dordrecht: Kluwer Academic Publishers, 1998. — P. 1–43.
3. **Brusilovsky P.** Efficient techniques for adaptive hypermedia // Lect. Notes. Comput. Sci. — 1997. — Vol. 1326 — P. 12–30.
4. **Brusilovsky P.** Methods and techniques of adaptive hypermedia // User Modeling and User-Adapted Interaction. — Vol. 6. — P. 87–129.
5. **De Bra P., Brusilovsky P., Houben G.-J.** Adaptive Hypermedia: From Systems to Framework // ACM Computing Surveys. — 1999. — Vol. 31, № 4.
6. **De Bra P.** Adaptive Hypermedia on the Web: Methods, techniques and applications // Proc. of the AACE WebNet'98 Conf. — Orlando, Fl., 1998. — P. 220–225.

М.П. Глуханков, П.А. Дортман, А.А. Павлов, А.П. Стасенко

ТРАНСЛИРУЮЩИЕ КОМПОНЕНТЫ СИСТЕМЫ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ SFP*

1. ВВЕДЕНИЕ

В настоящее время для проведения крупных научных расчетов все чаще используются различные суперкомпьютеры, имеющие параллельную архитектуру. Как правило, эти мощные машины расположены в вычислительных центрах, к которым программисты имеют доступ преимущественно по сети через FTP и Telnet-протоколы. Использование суперкомпьютера для отладки программ не является эффективным, к тому же вычислителю нет смысла привязываться к какой-то одной системе.

В Институте Систем Информатики СО РАН создается система функционального программирования SFP. Эта система позволит прикладным программистам создавать и отлаживать параллельные функциональные программы на обычном персональном компьютере, которые затем можно будет исполнять на супервычислителе. Более того, функциональный язык высокого уровня Sisal версии 3.0 [34], используемый в этой системе, позволяет в функциональные программы включать вставки на императивных языках. В настоящее время в SFP реализуется возможность вызова функций, написанных на языке Си. Язык Sisal предоставляет автоматическое управление параллелизмом, которое является результатом его функциональной семантики. В системе SFP Sisal-программы преобразуются в специально разработанные внутренние графовые представления, на которых проводятся различные оптимизирующие преобразования, производится интерпретация программ и которые транслируются в программы на языке Си. Эти представления хоть и называются внутренними, но есть возможность их просмотра при помощи системы Higres [32, 33]. Кроме того, предполагается создание специальной системы визуализации для предоставления программисту возможности просмотра процесса исполнения программы во время её интерпретации, а также управления процессом оптимизации.

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

В настоящей работе рассматриваются уже разработанные важнейшие части системы SFP, а именно: основанное на иерархических графах внутреннее представление программ, удобное для анализа и автоматического распараллеливания, блоки для синтаксического и лексического анализа, построения внутреннего представления, система тестирования SFP.

В разд. 2 представлена схема системы SFP, рассмотрено, какие этапы преобразований проходит исходная программа в этой системе. Разд. 3 кратко знакомит с языком программирования Sisal 3.0. Разд. 4 посвящен внутреннему представлению программ в системе. Разд. 5–8 представляют некоторые составляющие части системы. В разд. 9–12 рассматриваются проблемы тестирования системы SFP. И, наконец, в заключении представлены направления будущей работы.

2. СХЕМА РАБОТЫ СИСТЕМЫ SFP

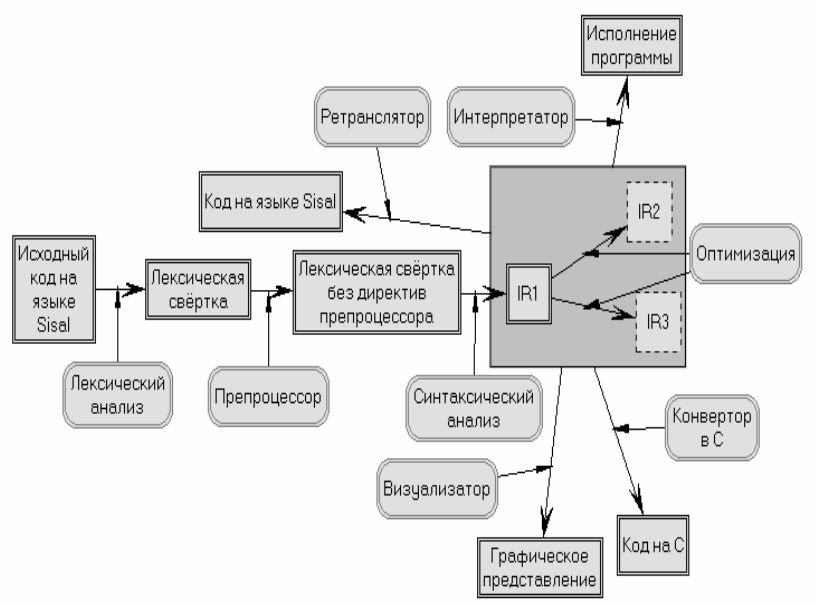


Рис. 1

На рис. 1 показаны основные части системы и их взаимодействие в процессе работы. Здесь прямоугольные вершины символизируют различные

уровни представления программ, т.е. то, с чем работают части системы. Закругленные вершины обозначают части системы, в результате работы которых представление программ переходит в новое состояние. Стрелочки между прямоугольными вершинами обозначают переход из одного состояния в другое, стрелочки, ведущие от закругленных вершин, обозначают, что соответствующий переход происходит под действием соответствующей функциональной части системы. Жирными рамками выделены те части и объекты системы, которые рассмотрены в настоящей работе.

Отметим, что вся работа программиста с системой происходит с помощью графического пользовательского интерфейса.

Рассмотрим жизненный цикл Sisal-программы в системе более подробно. Пользуясь текстовым редактором интерфейсной части, пользователь создаёт или загружает текстовый файл с исходным кодом. При попытке пользователя транслировать эту программу, под действием лексического анализатора происходит лексический анализ программы, и как результат в системе появляется лексическая свертка программы. Если в программе использовались директивы препроцессора, то они исполняются препроцессором уже на уровне лексической свертки, таким образом, после обработки препроцессором мы получаем лексическую свертку без директив препроцессора.

Внутреннее представление программ является важнейшим представлением программ в системе. Его напрямую использует большинство частей системы. Предполагается наличие трех видов внутреннего представления — IR1, IR2 и IR3. Отметим, что реально на настоящий момент разработано только IR1 представление, IR2 и IR3 будут разработаны позже. Представление IR1 служит для представления Sisal-программы в виде иерархического ациклического графа, представляющего поток данных в программе. В этом графе вершины соответствуют операциям, а дуги определяют передачу данных между вершинами, каждая дуга несет значение вполне определенного типа. Подробнее о внутреннем представлении см. разд. 4. Представление IR2 похоже на представление IR1, но в него добавлено распределение данных по памяти. Подобное представление IF2 описано в работе [28]. Над IR2 можно будет производить оптимизацию программ по памяти. Представление IR3 — это представление для императивных программ.

Система работает с внутренним представлением в зависимости от инструкций пользователя. Программист может просматривать внутреннее представление, используя систему визуализации графов Higres. Более того, используя графическое представление, пользователь может даже интерактивно изменять IR1-программу. Также пользователь может отлаживать IR1-

программу в режиме транслятора, и в целях отладки программа может быть ретранслирована из внутреннего представления в Sisal-программу. И, наконец, IR1-программа может быть переведена в программу на языке Си для компиляции и исполнения на суперкомпьютере.

3. ЯЗЫК SISAL 3.0

Sisal — язык функционального программирования, разработанный к 1983 г. в результате сотрудничества Манчестерского Университета, Ливерморской национальной лаборатории (США), Университета штата Колорадо и Digital Equipment Corporation [9]. В 1985 г. появилась версия 1.2 языка [10], для нее существует работающий оптимизирующий компилятор (OSC). В 1991 г. Ливерморская лаборатория и университет Колорадо опубликовали версию 2.0 языка [12, 14], которая после модернизации в 1995 г. получила название Sisal-90 [16, 17, 23], но так и не была никем реализована. В 2001 г. была представлена версия 3.0 языка Sisal [34]. В этой версии введены некоторые новые синтаксические конструкции, которые вносят в язык Sisal новые возможности, как, например, поддержка многоязыкового программирования (можно встраивать функции, написанные на языке Си), появились модульность и глобальные переменные.

Семантика языка Sisal имеет ряд важных свойств, которые благоприятствуют анализу и доказательству корректности программ. Во-первых, имена прозрачны для ссылок, имена переменных олицетворяют значения, а не ячейки памяти, а также нет прямой работы с памятью. Во-вторых, Sisal — язык однократного присваивания. В версиях языка до 2.0 включительно Sisal был математически правильным языком, т.е. функции не имели побочных эффектов. Но в двух последних версиях добавлены глобальные переменные. Версия Sisal 3.0 содержит такие конструкции, как функции высшего порядка, множества типов, массивы, потоки, союзы, записи, комплексные числа, включает в себя векторные операции над массивами, семантика которых аналогична семантике векторных операций языка Фортран-90. Кроме того, программы могут быть разбиты на модули.

4. ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ ПРОГРАММ (IR)

Внутреннее представление, являющееся системой Си++ классов, основано на представлении Sisal-программ в виде множества ациклических графов, с использованием идеи IF1 представления [26, 27]. Такие графы (точ-

нее было бы назвать их графовыми объектами) состоят из 4-х видов компонент: вершин, дуг, типов и границ графа. Вершины обозначают операции, такие как сложение и деление, дуги символизируют значения, передающиеся от вершины к вершине, типы могут быть закреплены за каждой дугой. Границы графа окружают группы вершин и дуг. Например, граф для выражения (X*X+Y*Y) представлен, как это показано на рис. 2.

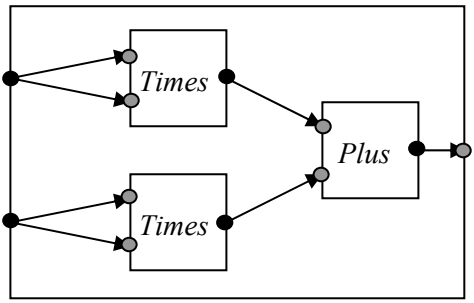


Рис. 2

Внешний прямоугольник обозначает границу графа. Этот граф имеет два порта для входных значений (*x* и *y*) и один выходной порт для результата. Прямоугольники внутри графа обозначают вершины. Все операции в примере имеют два входа и один выход, но, вообще говоря, число входящих и выходящих портов зависит от операции. Например, операция для построения массива может брать любое количество входных значений. Порты для каждой операции нумеруются и порядок нумерации важен.

Стрелочки в примере обозначают дуги. Дуги символизируют данные, передаваемые между вершинами (или между вершинами и границей графа). Дуги в рассматриваемом примере несут значения *x* и *y* внутрь графа, передают их квадраты из вершин *Times* в вершину *Plus* и отправляют результат за пределы графа. Дуги также несут информацию о типах данных, передаваемых по ним, но это не отражено на рисунке.

Каждую функцию программы мы пытаемся представить таким IR1-графом, и семантика языка Sisal способствует этому. Но простых вершин не всегда достаточно, для представления сложных языковых конструкций приходится использовать составные вершины.

Составных вершин не так много. Всего используются 5 типов составных вершин: для реализации альтернативы, выбора элемента из объединения и три типа для реализации различных циклов.

Таким образом, программа на языке Sisal представляется в виде конечного множества иерархических графов. Представление конкретной программы в виде IR1-графа мы называем IR1-программой.

Представление IR1 реализовано с помощью системы классов на языке Си++. Объектная модель очень удобна для представления таких графов. Реализованы интерфейсы для построения графов IR1, а также различные интерфейсы, необходимые для реализации анализа внутреннего представления и проведения над ним оптимизирующих преобразований. Объекты IR1 содержат много дополнительной информации для отладки, ретрансляции, интерпретации, оптимизации и других операций. За счёт добавления дополнительных интерфейсов объектная реализация может быть легко расширена до более сложной, позволяющей решать новые задачи без изменений уже реализованных частей системы.

5. ТРАНСЛЯТОР

В этом и следующих разделах описаны некоторые основные части системы SFP.

При рассмотрении трансляции языков функционального программирования существенным является этап синтаксического и семантического анализа, так как лексический анализ выполняется при помощи общих с анализом других языков методов.

Рассматривая интересующие нас этапы трансляции программ на языке Sisal 3.0, можно выделить несколько характерных особенностей его трансляции.

Во-первых, трансляция осуществляется в специальное графовое, ориентированное на манипуляцию с параллельными потоками данных, внутреннее представление IR1, что позволяет в значительной степени сохранить наглядное соответствие между конструкциями языка и объектами внутреннего представления.

Во-вторых, глобальная вложенность синтаксических конструкций языка гарантирует нам, что любая его конструкция имеет возвращаемые значения, что, в свою очередь, даёт нам возможность организовать синтаксический разбор концептуально в виде набора функций, разбирающих конкретную конструкцию языка и возвращающих её результаты вызвавшей её конструкции. Это приводит к более структурированной организации исходного кода транслятора, что повышает его общую читаемость, возможность внесения изменений и исправления ошибок.

В-третьих, конструкции на языке Sisal 3.0 строго типизированы, что позволяет уже на этапе внутреннего представления присвоить каждому потоку данных его тип, что, однако, несколько усложняет этап семантического анализа, но значительно упрощает дальнейшие преобразования внутреннего представления.

В-четвёртых, стоит отметить, что язык Sisal 3.0 не является “чистым” с точки зрения функционального программирования, так как он поддерживает операции присваивания. Но это не создаёт дополнительных проблем ввиду того, что все присваивания являются однократными, т. е. мы не можем, к примеру, присвоить значение какому-то отдельному полю записи, а вынуждены определять запись, указывая все её поля. Операция присваивания реализована в Sisal 3.0 достаточно интересным способом. Она может появляться только внутри специальной синтаксической конструкции let, которая в одной своей части содержит список присваиваний, свойственный императивным языкам, а в другой — набор выражений, использующий имена, определенные в первой части конструкции, количество которых определяет её результирующую арность.

6. КОНВЕРТОР ИЗ ВНУТРЕННЕГО ПРЕДСТАВЛЕНИЯ ЯЗЫКА SISAL 3.0 В ЯЗЫК СИ

При разработке системы функционального программирования на базе языка Sisal 3.0 одной из задач была возможность получать выходной код на языке Си. Эта задача ставилась и в более ранних версиях систем программирования Sisal. Построение конвертора позволило добиться необходимой переносимости кода, обеспечивая его компиляцию практически на всех известных в настоящий момент платформах. Более того, наличие такого конвертора позволило достичь понимания (пусть и преобразованного кода языка Sisal) практически любым программистом. Конвертор получает на вход внутреннее представление IR1-программы на языке Sisal, которая уже прошла проверку на грамматическую и синтаксическую правильность. При этом конвертор начинает построение готовой программы, строя функцию main, а затем производя последовательный разбор и построение выходного кода для всех вложенных в функцию main выражений. Разбор других функций, находящихся на одном уровне с main, производится аналогично разбору main. В дальнейшем, при обработке тела любой из функций (в IR1 они представлены в виде сложных вершин, отвечающих за объемлющие функции) происходит переход к обработке этих вершин. При этом типы пере-

менных переводятся в стандартные типы языка Си, что облегчает переносимость. Перевод осуществляется не в максимально эффективный код на Си, а в максимально похожий на свой исходный текст на Sisal, удобный для понимания программисту (с этой целью максимально сокращено введение новых имен, которые затрудняют понимание, а вновь создаваемые имена имеют понятную привязку к именам исходных файлов и функций). Кроме того, предусмотрена возможность использования в качестве внутреннего представления не IR1, а уже подвергшихся оптимизации IR2. Возможно также использовать вставки модулей языка Си. Внутреннее представление такого модуля является вершиной, имеющей лишь входные и выходные дуги, внутренности такого модуля — черный ящик, раскрываемый лишь на этапе компиляции Си-программы.

Хотелось бы отметить некоторые проблемы, возникшие при разработке конвертора, обусловленные тем, что Sisal является функциональным языком: подстановки функций, функции сокращения, глобальные переменные. Также хотелось бы отметить проблему с вызовами функций, решение которой предусматривает введение неявных дуг у функций (контекста). Кроме того, в этом конверторе найдены решения тех проблем, которых в силу синтаксиса не было в предыдущих версиях языка Sisal, таких как передача глобальных переменных, а также использование при вызове функций контекста. Конвертор реализован на языке Си++ в виде иерархии классов, которые производят соответствующее конструирование программ на Си.

7. ОТЛАДЧИК

Средства отладки системы функционального программирования позволяют интерактивно отлаживать параллельные Sisal-программы. К средствам отладки относятся: графический визуализатор графового внутреннего представления программ, ретранслятор внутреннего представления и сам отладчик. Все они интегрированы со средой разработки программ, легко запускаются и управляются из неё посредством нажатия кнопок на панели инструментов или запуска команд меню.

Отладчик представляет собой средство для пошагового исполнения Sisal-программ. Под шагом здесь понимается вычисление результирующих значений на всех выходных портах одной из вершин графового представления программы.

Отладчик позволяет просматривать промежуточные результаты вычислений на любом шаге, а также вести отчёт пошагового исполнения, в запи-

сях которого указываются вершины, их тип, соответствующие операторы, входные и выходные значения.

Перед началом каждого шага у пользователя (программиста) имеется возможность выбрать для него вершину среди тех вершин, у которых на всех входных портах вычислены значения. При этом выбор может быть осуществлён либо по средствам выбора вершины в окне графического визуализатора, либо с помощью выбора вершины из списка вершин-кандидатов, либо через указание соответствующего оператора в тексте программы. Для широкого использования последнего метода применяется ретранслятор внутреннего представления, так как после применения оптимизирующих преобразований соответствие между вершинами графового представления и исходной программой может быть утеряно.

Отладчик может выбирать вершину для вычисления на новом шаге и автоматически. При таком исполнении существует возможность визуализации процесса вычислений на графовом или текстовом представлении. Можно также просто просматривать типы вычисляемых операций, имена функций и результаты. Можно изменять скорость исполнения, указывая время простоя между шагами вычисления, устанавливать точки останова или указывать место в программе, до которого следует произвести вычисления и остановиться.

Есть возможность просмотра стека вызовов функций для каждой ветви исполнения.

Отладчик позволяет отлаживать программы до того, как написаны реализации некоторых функций. Если в программе встречается вызов такой функции, то отладчик может спросить результат её выполнения у программиста либо использовать значения по умолчанию. Этот же метод может быть использован для получения результатов вызовов функций, реализованных в других модулях или на других языках программирования.

Для запуска отладчика достаточно нажать одну из кнопок: “пошаговое исполнение программы с самого начала”, “исполнять до указанного места” или “исполнять программу в режиме отладки”. В последнем случае исполнение будет приостанавливаться в точках останова.

Во время интерпретирования отчёт об исполнении выводится в отдельное окно. Содержание отчёта программист может настраивать самостоятельно в зависимости от потребностей, нажав “кнопку настройка” отчёта. В окне настройки указывается: нужно ли выводить номер строки в программе для каждого выполняемого оператора, тип оператора, входные и выходные значения, имена функций в начале и в конце исполнения, список имён, для которых необходимо выводить значения, в момент их присваивания.

8. РЕТРАНСЛЯТОР

После построения транслятором IR1-представления, проведения над ним оптимизирующих преобразований (получения IR2 и IR3) может возникнуть потребность в восстановлении всего текста Sisal-программы или её участка по её графовому представлению. Для этой цели в SFP предусмотрена операция ретрансляции, которую выполняет ретранслятор. Данный инструмент может использоваться, например, при отладке Sisal-программы.

Входными данными для ретранслятора могут быть графы внутренних представлений IR1, IR2 или IR3. Результатом работы ретранслятора должен являться текст программы на языке Sisal, при повторной трансляции которой будет получено внутреннее представление эквивалентное исходному.

Структуры внутреннего представления делятся на две группы. Первая группа — это структуры, для которых есть полностью соответствующие операции языка. Вторая группа — структуры, для которых в языке нет аналогичных. Первые при ретрансляции переводятся непосредственно в свои аналоги, а вторые — в более сложные конструкции.

При ретрансляции возникает проблема именования выражений. Во-первых, во внутреннем представлении значения передаются между вершинами по дугам, и неизвестно, нужно ли именовать значение и, если нужно, то какое имя выбрать. Во-вторых, хотелось бы, чтобы отретранслированная программа была максимально похожа на исходную, а значит, имена в них должны по возможности совпадать. В-третьих, после проведения оптимизирующих преобразований соответствие с исходной программой может быть утеряно, и будет необходимо отказаться от одних имен, а другие — добавить.

В SFP для решения этой задачи во внутреннем представлении все объекты могут иметь имена. Транслятор задаёт эти имена в соответствии с исходной программой, а ретранслятор может использовать их для именования выражений в отретранслированной программе. Ретранслятор предоставляет настройку именования. Он может определить, каким выражениям необходимо дать имена, но при необходимости можно заставить его давать имена ещё и тем выражениям, у которых есть имена во внутреннем представлении. Можно также потребовать генерацию имён для абсолютно всех выражений. Ретранслятор либо генерирует имена, либо использует имеющиеся во внутреннем представлении. Для генерации имён применяется шаблон, который задаётся в окне настроек ретранслятора. Например, в имя функции можно добавить имя модуля, а в имя локального выражения можно включить имя функции.

Средство ретрансляции интегрировано со средой разработки программ, и ретрансляцию можно произвести в любой момент, если внутреннее представление уже построено, нажатием кнопки на панели инструментов. Во время отладки ретрансляция при необходимости выполняется автоматически (см. разд. 7).

9. ПОДХОД К ТЕСТИРОВАНИЮ SFP

Для тестирования транслятора языка Sisal 3.0 создана система тестов по методике первого покрытия [1–4]. В методике первого покрытия тесты проектируются на основе частично формализованной (ЧФ) модели текста документации. Основные положения метода можно сформулировать следующим образом.

Определим (неформально) комплект первого покрытия как комплект тестов, достаточно равномерно затрагивающий все аспекты и свойства языка программирования (ЯП). Простейший способ частичной формализации критерия (полноты) первого покрытия — потребовать, чтобы для каждого утверждения определенного вида, содержащегося в тексте документации, в комплекте нашелся хотя бы один нацеленный на проверку этого утверждения тест.

Это, конечно, весьма слабый критерий полноты (хотя бы потому, что проверяемые утверждения должны явно присутствовать в тексте документации, а не выводиться из него; к тому же каждое содержательное утверждение может допускать различные интерпретации в зависимости от контекста и т.п.).

Тем не менее опыт тестирования трансляторов показывает, что даже такой слабый критерий при применении к сложным ЯП требует, во-первых, серьезных затрат на создание комплекта, во-вторых, специальной организации работы, и за счет недостатков современной технологии создания трансляторов на практике оказывается достаточным для формирования комплектов тестов, способных находить серьезные дефекты реализаций ЯП.

Суть метода в систематическом сканировании текста стандарта с целью создания и сопровождения следующих файлов.

1. ЧФ-модель текста документации. Из текста документации выделяются и фиксируются в этой модели подлежащие проверке утверждения. Структура модели соответствует структуре документации. Такая модель играет роль перечня подозрений; по сравнению с

полным текстом документации она существенно короче и содержит лишь необходимую для данного вида тестирования информацию.

2. Комплект тестов (текущая версия). Каждое подозрение, зафиксированное в первом файле, детализируется в виде группы тестов. Уровень детализации определяется тестовиком, однако, учитывая, что перечень подозрений удовлетворяет относительно слабому критерию полноты и не может быть существенно улучшен в рамках данной методики, рекомендуется придерживаться принципа "один тест на подозрение".

Созданная система тестов содержит как корректные тесты по отношению к проверяемой документации, так и тесты, некорректные по отношению к проверяемой документации (цель — обнаружение дефекта в реакции на неправильные входные данные). Для каждого теста определёны условия верного результата прогона.

10. АВТОМАТИЗАЦИИ В ТЕСТИРОВАНИИ

Для автоматизации создания тестов и их прогона созданы три программы. Одна из них позволяет разбивать файл с большим количеством текстов тестов на отдельные файлы. В первой строке файла должно находиться окончание названий файлов тестов, например *.sse* — точка и расширение имён файлов. Далее идут тексты тестов, но перед каждым тестом должна стоять строка-разделитель, начинающаяся с ////. За строкой-разделителем должна идти строка с именем файла для следующего теста. Полное имя файла для теста получается конкатенацией этой и первой строк. После имени теста идёт сам текст теста, который заканчивается либо строкой-разделителем, либо концом файла.

Например, пусть имеется файл all.txt, содержащий следующее:

```
.sse
///////
empty
////
2comment
%comment1
%comment2
```

Если дать команду *cutter.exe all.txt* , то будут созданы два файла *empty.sse* и *2comment.sse* , первый — пустой, а второй будет содержать две строки:

%comment1
%comment2

Здесь cutter.exe — это и есть первая программа — консольное приложение Win32.

Вторая программа позволяет быстро просматривать тексты файлов и редактировать их. Это текстовый редактор под Windows, очень похожий на блокнот, дополненный двумя кнопками на панели инструментов: предыдущий и следующий. Если нажать на одну из этих кнопок, то редактор откроет, соответственно, предыдущий или следующий файл, записанный в той же папке и с тем же расширением, что и текущий редактируемый файл. При этом, если в текст были внесены изменения, то редактор спросит, нужно ли сохранить его. Фактически программа позволяет “листать” файлы и редактировать их.

Третья программа позволяет автоматически прогонять большое количество тестов и может генерировать тесты сама по шаблонам, она описывается в следующей главе.

11. ПРОГОН ТЕСТОВ

Программа для автоматической генерации и прогона тестов имеет встроенный текстовый редактор для редактирования файла с текстом шаблона и информацией для прогона набора тестов (этот файл будем называть документом, программа позволяет указывать документ). То, куда будут помещаться сгенирированные тесты и откуда будут браться тесты для прогона, определяется именем файла документа. И в текущей рабочей папке используется папка с тем же именем (только без расширения), что и имя документа. Документ может содержать две части, которые не являются обязательными, но должны идти в тексте документа строго в порядке: информация для прогона, шаблон. Если информация для прогона отсутствует, то прогон не будет осуществляться. Если отсутствует шаблон, то не будет осуществляться генерация тестов.

Информация для прогона может состоять из одной или нескольких строк вида

<r>что угодно (комментарий)

либо

<e>PathName ext,time;

либо

<t> PathName ext,time,code;

Первый вид используется для комментариев. Второй — для запуска программы с полным именем PathName, и параметром — именем файла с тестом (имена автоматически читаются с диска). При этом расширение имени файла с тестом заменяется расширением ext. После запуска программы наша тестирующая программа ждёт time секунд и уничтожает процесс запущенной программы, если он не закончен.

Третий вид используется почти так же, как и второй, но дополнен code — кодом возврата тестируемой программы, который она должна возвратить, чтобы тест считался пройденным.

Прогон осуществляется следующим образом. Для каждого файла из каталога с именем документа и расширением имени, указанным в первой строке, начинающейся с <e> или <t>, запускается указанная в этой строке программа в соответствии с видом строки. После того как эта программа завершит своё выполнение или по истечении тайм-аута ее процесс уничтожится, запускается программа из следующей строки, начинающейся с <e> или <t>, и т. д. Прогон теста заканчивается, если встречается строка, начинающаяся не с <r>, <e> или <t>. Затем из папки выбирается следующий файл с расширением имени, указанным в первой строке, начинающейся с <e> или <t>, и т. д.

Во время прогона тестов автоматически создаётся файл отчёта о прогоне, где указываются: имя папки с тестами, список произошедших ошибок и, если тестирование прервано, то пометка в конце: “ Прервано пользователем”. При описании ошибки указывается имя отказавшей программы, имя теста и тип ошибки (либо код возврата, либо превышение времени исполнения).

Данным средством можно обрабатывать текст теста последовательно несколькими программами, подавая на вход следующей результат предыдущей. Например, можно одной программой производить препроцессирование, второй — лексическую свёртку, а третьей — синтаксический анализ.

12. ШАБЛОНЫ ТЕСТОВ

Часто при создании тестов нужно написать множество тестов, очень похожих друг на друга. Например, при тестировании лексического анализатора может понадобиться подстановка в текст тестовой программы различ-

ных символов или ключевых слов. Для автоматической генерации такого рода тестов разработаны шаблоны текстов. Они являются частью документа приложения, описанного в предыдущей главе.

Шаблон идёт сразу после строк, начинающихся с <r>, <e> или <t>. Шаблон здесь — это текст, в котором могут встречаться теги. Тег — это текст, заключённый в угловые скобки: < , >. Чтобы в шаблоне поставить знак < , не считающийся началом тега, его нужно дублировать. Текст тега описывает набор значений. Для каждого значения тега будет сгенирирован тест. Тег можно записать в следующем виде <name=’file_name’> . Здесь *name* это имя тега, которое вместе со знаком ‘=’ можно опустить, file name — имя файла, строки которого будут значениями тега. Если в шаблоне присутствует лишь один тег, то будет создано столько тестов, сколько строк в файле file_name этого тега, при этом каждый тест будет получен из текста шаблона заменой тега на одно его значение. Если же тегов несколько, то для всевозможных комбинаций значений тегов будут созданы тексты тестов.

Имя нужно тегу для того, чтобы можно было в нескольких местах текста использовать одно и то же значение. Если имя тега определено, то далее в шаблоне можно указать тег вида <name> . Для того чтобы описать имя тега, а описание не было включено в тексты тестов, нужно использовать тег вида <*name=’file_name’> . Имя тега можно переопределять, тогда далее по тексту будет использоваться самое последнее определение.

Тесты могут создаваться не только для всевозможных комбинаций значений тегов, но и для значений, которые записаны в файлах file_name в строках с одинаковыми номерами. Для такого одновременного изменения значений нескольких тегов нужно после определения первого тега определить второй в виде <name=|’file_name’> или <*name=|’file_name’> . Вообще можно определять и более одного «паралельного» тега.

Пример.
Пусть есть шаблон

Function <fname=’names.txt’>(returns boolean)
*<’args1.txt’>+<arg2=|’args2.txt’><*res=|’results.txt’>+0*<’args3.txt’>=<res>*
End function %<fname>

и файлы:
Names.txt содержащий:

Main

Args1.txt содержащий:
11
12

Args2.txt содержащий:
2
1

Results.txt содержащий:
13
13

args3.txt содержащий:
1
2

Тогда будут сгенерированы файлы:

1.sse содержащий:
```
Function main(returns Boolean)
   11+2+0*1=13
end function %main
```

2.sse содержащий:
```
Function main(returns Boolean)
   12+1+0*1=13
end function %main
```

3.sse содержащий:
```
Function main(returns Boolean)
   11+2+0*2=13
end function %main
```

4.sse содержащий:
```
Function main(returns Boolean)
   12+1+0*2=13
end function %main
```

13. ЗАКЛЮЧЕНИЕ

На данный момент система SFP реализуется для работы на машинах под управлением системы Windows. Но коллектив разработчиков при реализации основных частей системы, таких как внутреннее представление, трансляторы, анализаторы, интерпретатор, кодогенератор использует только стандартные библиотеки Си++, поэтому в дальнейшем, при желании, основные компоненты системы можно будет перенести на другие платформы. Нужно будет реализовать в основном только интерфейс взаимодействия системы с пользователем.

В дальнейшем будет реализована возможность построения программ для супервычислителя (предполагается использовать RMI), а также расширение возможности встраивания текстов на других языках в программы на языке Sisal.

СПИСОК ЛИТЕРАТУРЫ

1. **Kaufman V., Pavlov M., Rybin S.** The Testing of Ada Compiler Diagnostics // ACM Ada Letters — 1993. — V.13, N4. — P. 71–76.
2. **Кауфман В.Ш., Рыбин С.И.** Методика контроля диагностики трансляторов // Программирование. — 1990. — № 4. — С.28–37.
3. **Рыбин С.И.** Методика тестирования диагностики нарушений контекстных условий в стандартизованных трансляторах: Автореф. дисс. канд. физ.-мат. наук. — М: МГУ, 1987.
4. **Гусляев А.М., Кауфман В.Ш., Рыбин С.И.** О проектировании диагностических тестов для Ада-трансляторов // Новые методы разработки проблемно-ориентированных программных систем. — Владивосток: ИАПУ ДВНЦ АН СССР, 1986. — С. 61–71.
5. **Allan S.J., Oldehoeft R.R.** Parallelism in SISAL: Exploiting the HEP architecture // Proc. of the 19th Hawaii Intern. Conf. on System Science. —1986, January. — P. 538–548.
6. **Bohm A., Sargeant J.** Efficient dataflow code generation for sisal // Tech. Rep. / University of Manchester — 1985.
7. **D.E. Culler et al.** Fine grain parallelism with minimal hardware support: A compiler-controlled Threaded Abstract Machine // Proc. of the 4th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems — April 1991.
8. **D. Cann.** Retire FORTRAN? a debate rekindled. — CACM — Aug. 1992. — Vol. 35, N 8 — P. 81–89.
9. **McGraw, J. R. et. al.** Sisal: Streams and iterations in a single assignment language, Language Reference Manual, Version 1.1 // Lawrence Livermore Nat. Lab. Manual M-146. — Livermore, CA — 1983.

10. **McGraw, J. R. et. al.** Sisal: Streams and iterations in a single assignment language, Language Reference Manual, Version 1.2 / Lawrence Livermore Nat. Lab. Manual M-146 (Rev. 1). — Livermore, CA 1985.
11. **Skedzielewski S. K., Welcome M. L.** Data flow graph optimization in IF1 // Lect. Notes Comput. Sci. — 1985. — Vol. 201. — P. 17–34.
12. **Cann D.C.** The optimizing SISAL compiler: Version 2.0. — Livermore — 1992. (Prepr. / Lawrence Livermore National Laboratory; UCRL-MA-110080).
13. **Cann D. C.** Compilation techniques for high performance high performance applicative compilation. — Colorado State, 1989. — (Tech. Rep. / Colorado State; NCS–89–108).
14. **Bohm A. P. W., Oldenhoeft R. R., Cann D. C., Feo J. T.** The SISAL 2.0 Reference Manual. — Livermore, CA, 1991. — (Prepr. / Lawrence Livermore Nat. Lab.; UCRL-MA-109098, LLNL).
15. **Евстигнеев В. А., Городняя Л. В., Густокашина Ю. В.** Язык функционального программирования SISAL // Интеллектуализация и качество программного обеспечения. — Новосибирск, 1994. — С. 21–42.
16. **Feo D. T., Miller P. J., Skedzielewski S. K., Denton S. M.** Sisal 90 User's Guide / Lawrence Livermore Nat. Lab. Draft 0.96. — Livermore, CA, 1995.
17. **Бирюкова Ю. В.** SISAL 90 руководство пользователя. — Новосибирск, 2000. — 84с. — (Препр./ Сиб. Отд-е. РАН ИСИ; № 72).
18. **Feo J. T.** Sisal. — Livermore, CA, 1992. — (Prepr. / Lawrence Livermore Nat. Lab.; UCRL-JC-110915, LLNL).
19. **Feo J. T., Cann D.C, Oldehoeft R.R.** A report on the Sisal language project // J. on Parallel and Distributed Computing. — 1990. — Vol. 10. — P. 349–366.
20. **McGraw J. R.** Parallel functional programming in Sisal: fictions, facts, and future. — Livermore, CA, 1993.— (Prepr. / Lawrence Livermore Nat. Lab.; LLNL).
21. **Kasyanov V. N., Evstigneev V.A. et al.** The system PROGRESS as a tool for parallelizing compiler prototyping // Proc. of Eighth SIAM Conf. on Parallel Processing for Scientific Computing (PPSC-97). — Minneapolis, 1997. — P. 301–306.
22. **Kasyanov V.N., Evstigneev V.A. et al.** Support tools for supercomputing and networking // Lect. Notes Comput. Sci. — 1999. — Vol.1593. — P. 431–434.
23. **Feo D. T., Miller P. J., Skedzielewski S. K., Denton S. M., Solomon C. J.** Sisal 90 // Proc. High Performance Functional Computing — Livermore, 1995. — P. 35–47.
24. **Трой Д.** Программирование на языке Си для персонального компьютера IBM PC / Пер. Б.А. Кузьмина под ред. И. В. Емелина. — М.: Радио и связь, 1991.
25. **Касьянов В. Н.** Оптимизирующие преобразования программ. — М.: Наука, 1988.
26. **Skedzielewski S. K., Glauert J.** IF1 — An intermediate form for applicative languages. Manual M-170 / Lawrence Livermore National Laboratory — Livermore, CA, 1985.
27. **Густокашина Ю.В., Евстигнеев В.А.** IF1 — промежуточное представление Sisal-программ // Проблемы конструирования эффективных и надежных программ. — Новосибирск, 1995. — С. 70–78.

28. **Welcome M., Skedzielewski S., Yates R.K., Ranelletti J.** IF2 — An applicative language intermediate form with explicit memory management / Lawrence Livermore Nat. Lab. Manual M-195. — Livermore, CA, 1986.
29. **Miller P. J.** TWINE: a portable, extensible SISAL execution kernel // Proc. Second SISAL User's Conf. — Livermore, 1992. — P. 243–256.
30. **Ranelletti J. E.** Graph transformation algorithms for array memory optimization in applicative languages. — Livermore, CA, 1987. — (Prepr. / Lawrence Livermore National Laboratory; UCRL-53832).
31. **Li Z., Kirkham C.** Efficient implementation of aggregates in united functions and obiects. — University of Manchester, 1995.
32. **Лисицын И.А.** Применение системы HIGRES для визуальной обработки иерархических графовых моделей // Проблемы систем информатики и программирования. — Новосибирск, 1999. — С. 64–77.
33. **Система** Higres. — Доступна по адресу: http://lis.iis.nsk.su/higres.
34. **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.

А. А. Дунаев, И. В. Лобив, Д. Ю. Мехонцев, Ф. А. Мурзин, О. Н. Половинко, Д. Ф. Семич, А. В. Чепель, К. А. Ярков

АЛГОРИТМЫ БЫСТРОГО ПОИСКА ФРАГМЕНТОВ ФОТОГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ*

ВВЕДЕНИЕ

Поиск фрагментов в изображениях — важная задача, которая в том или ином виде возникает во многих отраслях человеческой деятельности. В качестве примера можно привести подсчет количества бактерий на фотоснимке, поиск заданного участка местности на аэрофотоснимке и т. д.

Нередко задача осложняется тем, что искомый фрагмент может отличаться от образца расположением, цветовыми характеристиками, такими как яркость, контрастность, насыщенность цвета и др. Его геометрия может быть искажена, к примеру, вытянута по какому-то направлению.

Наконец, при разработке систем машинного зрения в робототехнике, систем безопасности, военных систем наведения и целеуказания возникает еще одно требование, состоящее в том, что поиск должен осуществляться максимально быстро.

На данный момент в мире существует всего несколько программных продуктов, осуществляющих быстрый поиск образца на изображении для случая, когда образец внутри данного изображения может быть повернут. Еще меньше программ, позволяющих выполнить поиск образца, искаженного аффинным преобразованием.

Один из таких продуктов — MaxVision Toolkit фирмы Datacube. Этот пакет представляет собой набор удобных в использовании, быстрых и высокоточных инструментов для машинного зрения, включающий процедуры калибровки съемочной аппаратуры, ввода, подготовки и анализа изображений и поиска объектов. При поиске осуществляется коррекция перспективного искажения. Для работы пакета требуются специализированные аппаратные средства. Более подробную информацию о фирме и ее продуктах можно получить на веб-сайте фирмы [9].

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

Существуют и другие программные и программно-аппаратные комплексы, осуществляющие поиск объектов теми или иными методами [10–12].

Целью данной работы являлось создание алгоритмов и программ, обнаруживающих заданный фрагмент на изображении. Образец для поиска и изображение, в котором осуществляется поиск, являются обычными образами, т. е. могут быть загружены из файлов в одном из распространенных растровых форматов. Искомый фрагмент может быть повернут, растянут или иметь яркость, контраст и насыщенность цвета, отличные от аналогичных параметров образца.

Реализованная программа состоит из двух частей — вычислительного ядра и управляющей оболочки, осуществляющей пользовательский интерфейс. Вычислительное ядро можно с минимальными изменениями, связанными с особенностями конкретной аппаратной платформы, использовать при разработке других программ. Программа разработана для операционной системы Microsoft Windows NT/2000.

Пользовательский интерфейс обеспечивает удобное управление функциями программы, вывод результатов в понятной форме, возможность скрыть ту информацию, которая временно не требуется пользователю. Поддерживается большинство широко используемых форматов файлов изображений.

1. ОПИСАНИЕ АЛГОРИТМОВ

1.1. Определения и обозначения

Изображение, которое требуется отыскать, будем называть образцом, а то, на котором производится поиск — исходным изображением.

При обработке изображений в программе используется система цветовых координат RGB.

Цветное изображение размером $n \times m$ задается тремя матрицами $S_R = S_R(i, j)$, $S_G = S_G(i, j)$ и $S_B = S_B(i, j)$, где $0 \le i \le n-1$, $0 \le j \le m-1$. Значения элементов матриц $S_R(i, j)$, $S_G(i, j)$ и $S_B(i, j)$ изменяются в пределах от 0 до 255.

Рассмотрим две точки на изображении $p = (i, j)$ и $p' = (i', j')$, имеющие цвета (r, g, b) и (r', g', b') соответственно. Это означает, что

$S_R(i,j)=r$, $S_G(i,j)=g$, $S_B(i,j)=b$, $S_R(i',j')=r'$, $S_G(i',j')=g'$, $S_B(i',j')=b'$. Для краткости будем писать $S_R(p)$ вместо $S_R(i,j)$, $S_G(p)$ вместо $S_G(i,j)$ и т.д.

Определим цветовое расстояние между точками:

$$cd(p,p')=\max\{|S_R(p)-S_R(p')|,|S_G(p)-S_G(p')|,|S_B(p)-S_B(p')|\}. \quad (1.1)$$

Евклидова метрика определяется следующим образом:

$$\rho(p,p')=\sqrt{(i-i')^2+(j-j')^2}\ . \quad (1.2)$$

В процессе сканирования изображения, а также после некоторых преобразований, например, после фильтрации, некоторые цвета могут измениться. Цвета, которые прежде были идентичны, станут различными. В то же время обычно эти цветовые изменения невелики. Поэтому в дальнейшем мы будем использовать специальную константу, которая называется цветовой константой и обозначается C_V. Цветовая константа определяет порог цветового расстояния между двумя цветами, ниже которого эти цвета считаются идентичными, т.е. если

$$cd(p,p')\le C_V\ ,$$

то считается, что точки p и p' имеют один и тот же цвет.

Будем обозначать через $B_n(p)=B_n(i,j)$ квадрат размером $n\times n$ с центром в точке $p=(i,j)$, где n — нечетное.

Перед началом работы алгоритма поиска производится фильтрация некоторых шумов. Существует большое количество различных методов фильтрации, направленных на выделение или подавление тех или иных свойств сигнала. Это направление в обработке сигналов хорошо изучено [5–8]. В настоящей работе изображение подвергается сначала низкочастотной, а затем — медианной фильтрации.

1.2. Низкочастотная фильтрация

Целью применения двумерной низкочастотной фильтрации является получение изображения, имеющего более гладкие цветовые характеристики, чем исходное. Это нужно для того, чтобы сделать процесс поиска изображения более устойчивым. Для каждой точки $p=(i,j)$ вычисляется

среднее арифметическое яркостей в окрестности $B_n(p)$, где n — нечетное, т.е.

$$S'_{R,G,B}(i,j) = \frac{1}{n^2} \sum_{u=-\frac{n-1}{2}}^{\frac{n-1}{2}} \sum_{v=-\frac{n-1}{2}}^{\frac{n-1}{2}} S_{R,G,B}(i+u, j+v). \qquad (1.3)$$

1.3. Медианная фильтрация

Медианная фильтрация состоит в том, что в окрестности каждой точки изображения выбирается точка, значение цвета которой является средним в упорядоченной последовательности значений цветов всех точек окрестности, причем по каждой компоненте цвета фильтрация производится отдельно. Иными словами, мы вычисляем $p^* = (r^*, g^*, b^*)$, где r^* — это средний элемент в упорядоченной по возрастанию последовательности значений компоненты r всех точек окрестности, а g^* и b^* — соответственно g и b. Медианная фильтрация подавляет импульсные помехи.

Одновременно с подавлением импульсных помех удаляются точки с высокочастотными цветовыми характеристиками. Поставим в соответствие с точкой p некоторую ее окрестность $B_n(p)$, где n — нечетное. Тогда, если $cd(p, p') \le C_V$ для достаточно большого количества точек $p' \in B_n(p)$, что задается дополнительной константой, например, от 7 до 9 точек для окрестности 3×3 или от 20 до 25 точек для окрестности 5×5, то полагаем $S'(p) = (r^*, g^*, b^*)$ и $f(i,j) = 0$. В противном случае $S'(p) = (r^*, g^*, b^*)$ остается прежней и $f(i,j) = 1$. Характеристическая функция $f(i,j)$ показывает, подходит ли точка (i,j) для дальнейшего рассмотрения; она принимает значение 0 в точках, имеющих высокочастотные цветовые характеристики, и значение 1 в точках, не имеющих таких характеристик.

1.4. Выделение контуров

Пусть p', p'' — две точки исследуемого изображения,

$$S(p') = (r', g', b'),\ S(p'') = (r'', g'', b'').$$

Пусть теперь

$$cd[B_m(p)] = \max\{cd(p', p'') : p', p'' \in B_m(p)\} \quad (1.4)$$

— максимальное цветовое расстояние в окрестности $B_m(p)$ исследуемой точки. Теперь

$$f(p) = \begin{cases} 0, \ if \ cd[B_m(p)] \geq C_V, \\ 1, if \ cd[B_m(p)] < C_V. \end{cases}$$

— если цветовое расстояние превышает заданный порог, то через рассматриваемую точку проходит контур. Заключение о том, проходит контур через точку или нет, фиксируется соответствующими значениями функции f (0 или 1 соответственно).

1.5. Дополнительные условия преобразований

Описанные преобразования — как фильтрация, так и выделение контуров, — производятся как для изображения-образца, так и для исходного изображения.

1.6. Выбор опорных точек

Вопрос выбора точек, которые будут опорными, является интересным и достаточно глубоко изученным [1–4]. При выборе точек можно учитывать их яркостные характеристики. Также могут использоваться анализаторы топологических свойств изображения. Возможно применение градиентных или пространственно-частотных методов.

В настоящей работе используется эвристический метод, который при испытаниях показал хорошие результаты. Этот метод был предложен Д. Мехонцевым (Институт математики СО РАН), он является одним из вариантов эвристических методов, используемых другими авторами (В. Люцив, В. Малышев, А Можейко, Государственный оптический институт). Выбираются точки, максимально удаленные друг от друга в специальной метрике (правило выбора описано ниже), которая является линейной комбинацией цветовой и евклидовой метрик:

$$\mathrm{P}(p, p') = A \cdot \sqrt{(i - i')^2 + (j - j')^2} + B \cdot cd(p, p'), \quad (1.5)$$

где A и B — константы, в общем случае не зависящие от свойств конкретного изображения. Значения этих констант, при которых поиск наиболее устойчив, можно подобрать опытным путем.

Алгоритм выбора опорных точек работает следующим образом. Сначала изображение сканируется слева направо и сверху вниз. Первая точка контура, встретившаяся при сканировании, выбирается в качестве первой опорной точки S_1. После этого выбирается точка S_2, также лежащая на контуре и при этом максимально удаленная от S_1 в метрике (1.5), затем — точка S_3, лежащая на контуре и при этом максимально удаленная от S_1 и S_2 в метрике (1.6), и так далее до тех пор, пока не будет выбрано требуемое количество опорных точек.

1.7. Построение структуры вспомогательных данных

Для ускорения процесса поиска опорных точек на исходном изображении перед поиском генерируется специальная структура вспомогательных данных, ассоциированная с исходным изображением. На ее построение затрачивается время, пропорциональное размеру изображения $n \times m$, после чего появляется возможность находить опорные точки за логарифмическое время, т.е. за $C \cdot \log(n \times m)$ шагов, где C — константа, характеризующая время выполнения одного шага поиска и не зависящая от размеров исходного изображения.

Ниже описывается процесс построения деревообразной поисковой структуры. Исходное изображение последовательно делится на равные прямоугольники:

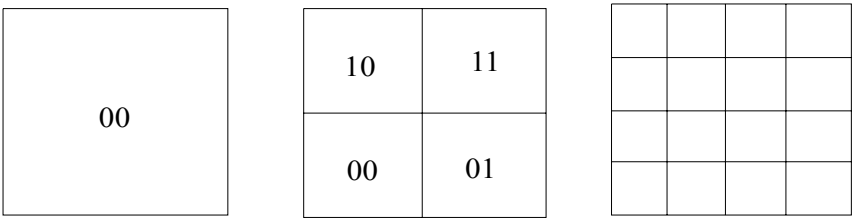


Рис. 1. Разбиение изображения на прямоугольники

Каждое последующее разбиение является более мелким, чем предыдущее. На первом шаге не имеется никакого разбиения. Последнее разбиение состоит из прямоугольников, размер которых равен одному пикселю. Предыдущее (более грубое) разбиение в этой последовательности конструируется из следующего (более тонкого). Прямоугольники, которые содержат данное разбиение, могут быть занумерованы парами целых чисел.

Заметим также, что совокупность всех этих разбиений можно представить в виде дерева, в корне которого располагается самое грубое разбиение, и следующим уровнем для каждого разбиения являются разбиения следующего порядка, на которые оно может быть разложено.

Пусть (i, j) — пара, используемая для нумерации прямоугольника. Обозначим через $R_{r,s}(i, j)$ соответствующий прямоугольник размера $r \times s$.

Теперь опишем метод построения грубых палитр. Значения r, g, b для любой точки представляются посредством 8 бит. Пусть $0 < C_r \le 8$, $0 < C_g \le 8$, $0 < C_b \le 8$, где C_i — компоненты цветовой константы.

Обозначим $\bar{r}$ байт, у которого старшие C_r бит $\bar{r}$ равны соответствующим значениям бит в r, младшие $8 - C_r$ бит $\bar{r}$ равны 0. Аналогично определим значения $\bar{g}$ и $\bar{b}$. В общем случае такая палитра содержит $2^{C_r} \times 2^{C_g} \times 2^{C_b} = 2^{C_r + C_g + C_b}$ цветов. Обозначим множество всех этих цветов

$Pal(C_r, C_g, C_b)$. Таким образом, мы построили грубую палитру, которая не содержит цветов, расстояние между которыми меньше цветовой константы.

Далее с $R_{r,s}(i,j)$ ассоциируется определенная на рассматриваемой палитре характеристическая функция $\chi_{k,ij}$. Эта функция зависит от узла в дереве. Узел характеризуется тройкой (k,i,j), где k — уровень в дереве и (i,j) — пара, индексирующая данный прямоугольник. Аргументы χ_{kij} суть тройка $(\widetilde{r},\widetilde{g},\widetilde{b})$ цветов грубой палитры. Функция χ_{kij} показывает, существует ли точка, имеющая цвет (r,g,b), в прямоугольнике $R_{r,s}(i,j)$ на уровне k, который является близким к цвету (r,g,b).

Пусть $(\widetilde{r},\widetilde{g},\widetilde{b}) \in Pal(C_r, C_g, C_b)$. Определим $\chi_{kij}(\widetilde{r},\widetilde{g},\widetilde{b})$ следующим образом: $\chi_{kij}(\widetilde{r},\widetilde{g},\widetilde{b}) = 1$, если существует такая точка $p \in R_{r,s}(i,j)$, что $S(p) = (r,g,b)$, и для некоторых r', g', b' таких, что $|r - r'| \le C_V$, $|g - g'| \le C_V$, $|b - b'| \le C_V$, выполнено $\overline{r'} = \widetilde{r}$, $\overline{g'} = \widetilde{g}$, $\overline{b'} = \widetilde{b}$.

В противном случае $\chi_{kij}(\widetilde{r},\widetilde{g},\widetilde{b}) = 0$.

Множество связанных с каждым узлом дерева характеристических функций $\chi = \{\chi_{kij} : 0 \le i \le \ldots, 0 \le j \le \ldots\}$ является той самой специальной структурой данных, упоминавшейся выше.

Функция, соответствующая разбиению A, может быть построена дизъюнкцией функций, соответствующих более точным разбиениям, на которые разбивается A.

1.8. Поиск опорных точек

Алгоритм поиска опорных точек работает следующим образом.

Шаг 1. Проверяем, существуют ли точки на исходном изображении, имеющие те же самые цвета, что и опорные точки, с точностью до грубой палитры.

Шаг 2. Рассматриваем более тонкое разбиение и пытаемся найти распределение опорных точек внутри соответствующих прямоугольников, принимая во внимание расстояния между опорными точками.

С каждым шагом мы рассматриваем все более мелкие разбиения. Это может быть сделано рекурсивно. Мы обходим описанное в предыдущем разделе дерево, описывающее вложения прямоугольников друг в друга.

Использование более и более мелких разбиений приведет к фиксации опорных точек.

Например, позиция точки p_1 может быть охарактеризована последовательностью чисел 00, 00, 11, 01, в которой числа индексируют прямоугольники на каждом следующем уровне дерева, в которых находится точка p_1. Мы должны обойти все такие последовательности, принимая во внимание дополнительную информацию (цвета и расстояния), и выбрать подходящую.

Переходя к более тонкому разбиению, мы находим более точные возможные позиции опорных точек. При этом возможны различные методы оптимизации этого процесса.

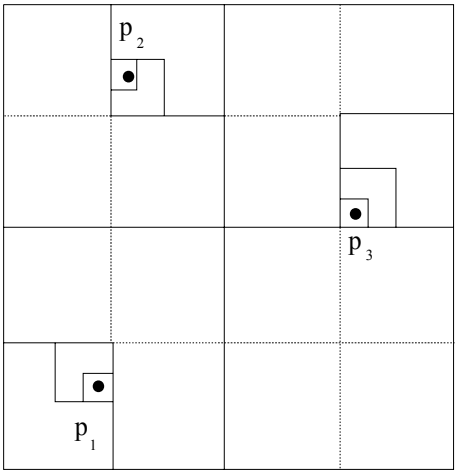


Рис. 2. Итерационный процесс фиксации опорных точек

Опишем алгоритм более детально. В любой момент мы работаем на некотором уровне разбиения, имеющем номер i. Равенство $i = 0$ обозначает, что мы имеем наиболее тонкое (пиксельное) разбиение. Число i возрастает при переходе от более тонкой к более грубой сетке.

Предположим, что выбраны некоторые прямоугольники, являющиеся подходящими для возможного расположения опорных точек $p_0, p_1, p_2, \ldots, p_k$.

Количество опорных точек, рассматриваемое на данном уровне, извлекается из массива max_point, который содержит количества опорных точек для каждого уровня.

Далее, обозначим k — номер последней найденной опорной точки. По ходу работы алгоритма, как только найдена очередная опорная точка, значение k увеличивается на 1. Как только k достигает max-point[i], т.е. последняя точка вписана в какой-то прямоугольник, мы переходим к более тонкой сетке, т.е. на уровень $i-1$.

На более тонкой сетке уточняется положение опорных точек с самого начала, т.е. с точки p_0. Отыскивается прямоугольник возможного расположения точки p_{k+1} на данном уровне i. Заметим, что поиск новых прямоугольников производится внутри прямоугольников, фиксированных на предыдущем уровне.

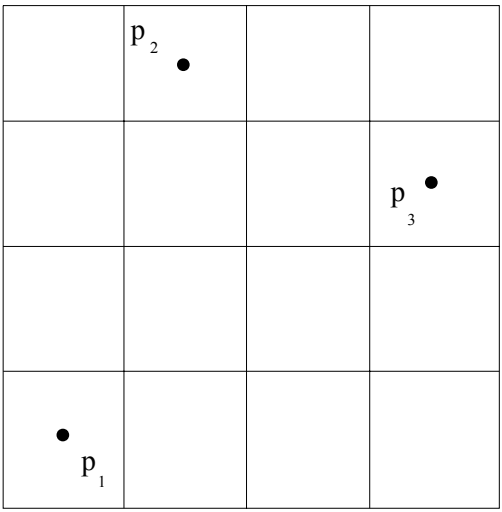


Рис. 3. Фиксация прямоугольников на данном уровне

Алгоритм поиска прямоугольника возможного расположения точки p_{k+1} выполняется следующим образом.

Предположим, что прямоугольники $Q_0,\cdots Q_k$ уже фиксированы. Рассмотрим новый прямоугольник Q_{k+1}. Введем параметры lp и $ps[i]$. Параметр ps будет зависеть от уровня.

Полагаем $lp = 0$, когда Q_{k+1} не содержит подходящего цвета. Этот случай не интересен, поскольку наличие цвета в прямоугольнике проверяется тривиальным образом.

Далее, $lp = 0$, если Q_{k+1} содержит подходящий цвет, но не соответствует по расстоянию между данным Q_{k+1} и Q_0. Аналогично $lp = 1$, если Q_{k+1} содержит подходящий цвет, Q_{k+1} имеет подходящее расстояние между данным Q_{k+1} и Q_0, но неподходящее расстояние между данным Q_{k+1} и Q_1.

В общем случае lp равно максимальному t такому, что Q_{k+1} имеет подходящее расстояние между Q_{k+1} и $Q_0,\cdots Q_{t-1}$, но неподходящее расстояние между Q_{k+1} и Q_t.

Таким образом, мы рассматриваем все возможные позиции для Q_{k+1}. Если подходящая позиция не найдена, то мы сдвигаем прямоугольник Q_{lp}. Это означает, что $Q_0,\cdots Q_{lp}$ выбраны корректно. Процесс продолжается, но мы ищем новую позицию для Q_{lp}. Записываем k в параметр ps в случае, когда $ps < k$. В самом начале полагаем $ps = 0$.

Мы можем получить ситуацию, когда процесс нахождения новых опорных точек не может быть продолжен на данном уровне. Тогда мы должны подняться на предыдущий уровень. В этом случае оказывается, что $lp = 0$, и нет никакой возможности найти новое расположение для p_0. Это возможно из-за того, что прямоугольники найдены внутри прямоугольников, фиксированных на предыдущем уровне, но нужные позиции не существуют на данном уровне. Тогда мы поднимаемся на предыдущий уровень и сдвигаем прямоугольник Q_{ps}, т.е. мы предполагаем, что позиции $p_0,\ldots,p_{ps-1}$ найдены корректно.

Из определения ps, данного выше, легко видеть, что этот параметр фиксирует максимальное число точек, которые были размещены успешно. Но параметр ps используется только в том случае, когда был осуществлен возврат на более грубую сетку.

Добавим, что мы должны учитывать не только расстояния между точками, но и их ориентацию, чтобы исключить отражения. Это может быть сделано проверкой знака соответствующего определителя

$$sign\begin{vmatrix} i_2 - i_1 & j_2 - j_1 \\ i_3 - i_2 & j_3 - j_2 \end{vmatrix},$$

где индексы у компонент соответствуют номерам точек, ориентация которых исследуется.

1.9. Поворот изображения

При финальном сравнении может оказаться, что образец, найденный на исходном изображении, повернут относительно его исходного положения. В этом случае применяется алгоритм поворота найденного участка исходного изображения, затем повернутый фрагмент сравнивается с образцом. Алгоритм работает следующим образом. На исходном изображении вычисляются координаты углов фрагмента, который необходимо повернуть.

Цвет пикселя результирующего изображения с координатами (i, j) устанавливается равным цвету пикселя исходного изображения, в который попал угол пикселя повернутого фрагмента с координатами (i, j). Искажения, вносимые таким алгоритмом поворота, слабо влияют на устойчивость поиска, поскольку на стадии финального сравнения используются фильтрованные изображения. Фильтры, как уже отмечалось ранее, придают изображениям необходимую гладкость и подавляют импульсные помехи, которые при искажениях, привнесенных при повороте, дали бы заметное изменение разности сравниваемых изображений.

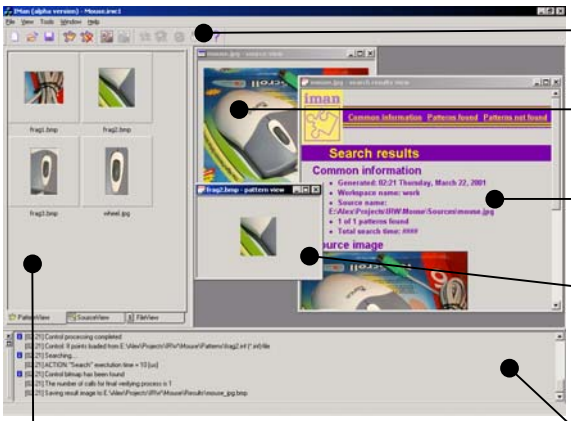
2. ОПИСАНИЕ РЕАЛИЗАЦИИ ПРОГРАММЫ

2.1. Общая схема реализации и интерфейс

Программа состоит из двух основных блоков, которые выполняют принципиально разные задачи. Первый блок реализует пользовательский интерфейс, будем в дальнейшем называть его интерфейсным блоком или просто интерфейсом. Второй блок выполняет подготовку изображений и их обработку.

Интерфейсная часть программы спроектирована по смешанной схеме; хотя одновременно может быть открыто не более одного документа, внешне программа выглядит типичным для многодокументных приложений образом (рис. 4). В нашем случае документом является так называемый проект, в котором хранится информация о файлах используемых исходных изображений и изображений-образцов.

В приложение встроен мастер создания нового проекта, позволяющий задать имя нового проекта и путь к папке, в которой этот проект будет создан. Мастер также позволяет задать один из двух режимов работы с добавляемыми в проект файлами изображений. В одном режиме файлы изображений копируются в папку проекта, а в проект добавляются только имена файлов, в другом режиме файлы не копируются, а в проект добавляются абсолютные пути к ним.



Панель инструментов.

Вид исходного изображения.

Вид результатов поиска.

Вид образца

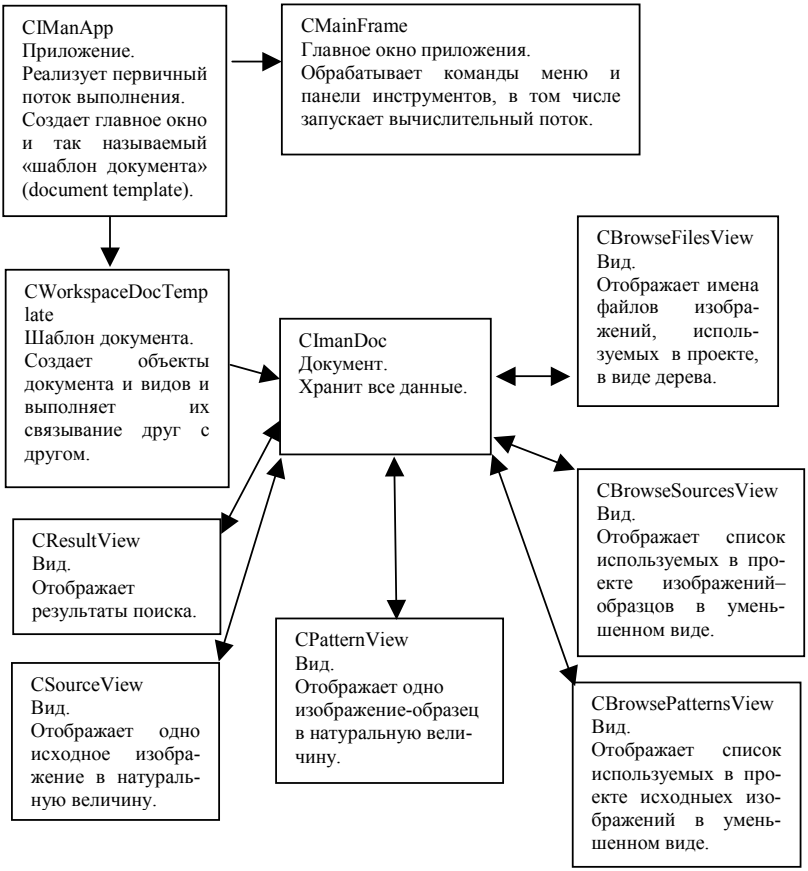
Виды предпросмотра исходных изображений, предпросмотра образцов и структуры проекта объединены в единую панель проекта, выбор нужного вида осуществляется выбором закладки внизу панели

Журнал событий

Рис. 4. Главное окно программы

Главное окно приложения позволяет выбрать исходное изображение и образцы для поиска, а затем выполнить поиск.

Результаты поиска сохраняются в формате HTML. Для масштабирования изображений используется библиотека NexgenIPL (свободно распространяемый продукт компании Binary Technologies, [13]). В процессе разработки классов панели проекта и журнала событий использованы некоторые решения, доступные на веб-сайте Codeguru, посвященном программированию [14].



В приложение встроены две утилиты, облегчающие поиск графических файлов на локальных дисках. Первая утилита отображает дерево папок выбранного локального диска, и пользователь имеет возможность открыть нужную ему папку и добавить в проект находящиеся в ней графические

файлы. Вторая утилита осуществляет поиск файлов по маске и выводит список файлов, найденных на заданном локальном диске, с тем чтобы пользователь мог добавить файлы в проект из списка найденного.

Интерфейс программы разработан с использованием библиотеки классов MFC в соответствии с идеологией «документ-вид».

Вычислительный блок реализован в виде набора классов, предназначенных исключительно для хранения данных, и нескольких процедур, выполняющих операции над ними. Вся работа с этими процедурами выполняется в специальной функции, запускаемой в отдельном потоке, приоритет которого устанавливается ниже, чем приоритет главного потока.

Теперь коротко остановимся на программной реализации фильтров. Везде ниже по тексту переменные m и n обозначают соответственно ширину и высоту изображения.

2.2. Реализация низкочастотного фильтра

Алгоритм двумерной низкочастотной фильтрации реализован стандартным для методов рекурсивной фильтрации образом, поэтому описание его реализации вынесено в Прил. 1.

2.3. Реализация медианного фильтра

Алгоритм медианной фильтрации так же, как и алгоритм двумерной низкочастотной фильтрации, реализован стандартным образом; описание реализации см. в прил. 1.

2.4. Выделение контуров

Выделение контуров выполняется точно так, как описано в п. 1.4, т.е. при каждой итерации алгоритма вычисляются цветовые расстояния для всех возможных пар точек исследуемой окрестности, затем выбирается максимальное значение расстояния.

2.5. Формирование вспомогательных данных

В программе функция χ_{kij} реализована с помощью подпрограммы-функции, обращение к которой имеет вид

```
this_color(level k, coordinate i, coordinate j, color RGB, int
                                    position, int mask)
```

Первые три параметра являются координатами, четвертый параметр задает цвет, который требуется найти. Объясним значения остальных двух параметров, position и mask.

Поскольку грубая палитра содержит $2^{C_r+C_g+C_b}$ цветов, для хранения значений функций χ_{kij} достаточно использовать $2^{C_r+C_g+C_b}/32$ слов длиной в 32 бита. Например, если $C_r = C_g = C_b = 4$, то достаточно использовать 128 32-битных слов.

Тройка $(\bar{r},\bar{g},\bar{b})$ может быть представлена в бинарной форме как конкатенация бинарных слов $\bar{r}$, $\bar{g}$, $\bar{b}$. Таким образом, мы получаем одно бинарное число вместо трех цветов. Обозначим это число bin. Тогда position = bin/32.

Значение этого числа следующее. Мы имеем бинарное слово длины $2^{C_r+C_g+C_b}$, разбитое на блоки длиной 32 бита. Значение position показывает, в каком блоке расположено значение bin.

При использовании структуры в поисковом процессе мы можем найти необходимый бит с помощью конъюнкции 32-битных слов с подходящей маской, имеющей форму $0\ldots010\ldots0$. Эта маска передается параметром mask.

2.6. Поиск опорных точек

Заметим, что в процессе поиска мы имеем дело с прямоугольниками, а не с точками.

Предположим, что мы имеем прямоугольники на некотором уровне дерева. Обозначим $d_{\min}$ и $d_{\max}$ — минимальное и максимальное расстояния между ними.

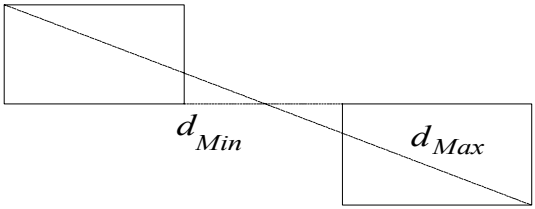


Рис. 5. Минимальное и максимальное расстояния между прямоугольниками

Пусть изображение S разделено на равные прямоугольники Q_{ij}. Всевозможные минимальные и максимальные расстояния между ними хранятся в виде двумерной таблицы расстояний. Более точно минимальные и максимальные расстояния от Q_{00} (прямоугольника с номерами 00) до Q_{ij} (прямоугольника с номерами *ij*) располагаются в позиции таблицы, имеющей номера *ij*. Расстояние между произвольными прямоугольниками может быть вычислено с помощью параллельного переноса одного из них в начало координат. Если левые нижние углы прямоугольников имеют координаты (i_1, j_1) и (i_2, j_2) соответственно, то после переноса координаты одного из них станут $(0,0)$, а другого

$$i_{new} = | i_1 - i_2 |, \; j_{new} = | j_1 - j_2 | .$$

Таким образом, расстояния могут быть взяты из той же самой таблицы расстояний.

Расстояния между опорными точками на образце известны. В процессе поиска опорных точек на исходном изображении приближенно проверяются расстояния между прямоугольниками на исходном изображении:

$$d_{\min} - pp \le d \le d_{\max} + pp ,$$

где d — известное расстояние на образце, а pp — допустимый дефект. В таблице расстояний хранятся $d_{\min} - pp$ и $d_{\max} + pp$, т.е. принимается во внимание дефект.

2.7. Организация обратной связи от вычислительного блока к интерфейсу

Передача информации от вычислительного блока к интерфейсу выполняется посредством сообщений Windows. Большинство функций ядра отправляют интерфейсу сообщения, которые заносятся в журнал событий.

ЗАКЛЮЧЕНИЕ

В результате проведенных исследований создан ряд алгоритмов, которые предназначены для быстрого поиска внутри данного изображения образцов, повернутых на некоторый угол и имеющих другой масштаб.

В алгоритме поиска используется метод опорных точек. На изображении-образце специальным образом выбираются опорные точки, которые затем отыскиваются на исходном изображении.

Результаты исследований могут быть применены в следующих областях:

- спутниковое объектное распознание, аэрофотосъемка и картографирование;
- для поиска в архиве фотографических изображений. Это могут быть фотографии кристаллической структуры металла, биологических материалов, полученных с помощью микроскопа и т.д.;
- на поточной линии; алгоритм может выбирать детали и действовать в системе машинного зрения для автоматизированных сортировочных машин, т.е. он может использоваться на сложных роботизированных производствах.

Реализована программа Iman, которая берет источник и список образцов, выясняет, включен ли каждый образец полностью в источник и устанавливает угол поворота для образцов, которые включены. Результаты поиска помещаются в отчет работы программы, который генерируется в виде HTML-файла, и может быть просмотрен с использованием внутреннего средства просмотра сообщений программы IMan или в броузере сети.

Разработанный алгоритм является одним из лучших в мире и вызывает большой интерес у японских специалистов в связи с применением в робототехнике. Главное преимущество нового подхода состоит в том, что все образцы разыскиваются одновременно. В типичных ситуациях, интересных с точки зрения приложений в робототехнике, время поиска образца измеряется миллисекундами.

СПИСОК ЛИТЕРАТУРЫ

1. **Братцев С. Г., Мурзин Ф. А., Нартов Б. К., Пунтус А. А.** Конфликт сложных систем. Модели и управление. — М.: Изд. МАИ, 1995.
2. **Братцев С. Г., Мурзин Ф. А., Нартов Б. К.** Исследования по обработке динамических изображений // Тез. междунар. конф. по обработке изображений и дистанционным исследованиям. — Новосибирск, 1990. — С. 41–43.
3. **Bratsev S. G., Murzin F. A., Nartov B. K.** The optimum search of targets and the processing of dynamical images // Proc. of the Intern. Symp. on Visual Analysis and Interface. — Novosibirsk, 1991. — P. 17.
4. **Грицык В. В.** Распараллеливание алгоритмов обработки информации в системах реального времени. — Киев: Наумова думка, 1981.

5. **Писаревский А.Н. и др.** Системы технического зрения. — М.: Машиностроение, 1988.
6. **Прэтт У.** Цифровая обработка изображений. — М: Мир, 1982. — Т. 1–2.
7. **Обработка** изображений / Под ред. Хуанга. — М.: Мир, 1979.
8. **Голд Б., Рэйдер Ч.** Цифровая обработка сигналов. — М.: Советское радио, 1973.
9. **Datacube**, Inc. `http://www.datacube.com`
10. **Media** Cybernetics, L. P. `http://www.optimas.com`
11. **Ronald A.** Massa Associates `http://www.way2c.com`
12. **Impuls** GmbH `http://www.impuls-imaging.com`
13. **Binary** Technologies `http://www.binary-technologies.de`
14. **Codeguru** `http://www.codeguru.com`

Приложение 1

РЕАЛИЗАЦИЯ ФИЛЬТРОВ

Реализация низкочастотного фильтра

Описание дано для фильтра с окрестностью размером 3×3.

Сначала для всех $0 \le j \le m-1$ вычисляются частичные суммы:

$$\sigma_R(j)=\sum_{i=0}^{2} S_R(i,j)\ ,\ \sigma_G(j)=\sum_{i=0}^{2} S_G(i,j)\ ,\ \sigma_B(j)=\sum_{i=0}^{2} S_B(i,j)\ .$$

Иными словами, суммируются яркости (отдельно для каждого цвета) по столбцам высотой 3 для всех столбцов внутри слоя, образованного тремя верхними рядами точек.

Вычислив $\sigma_R(0),\sigma_R(1),\sigma_R(2)$ и т.д., получим равенства:

$$S'_R(1,1)=\frac{1}{9}\sum_{k=0}^{2}\sigma_R(k)\ ,\ S'_G(1,1)=\frac{1}{9}\sum_{k=0}^{2}\sigma_G(k)\ ,\ S'_B(1,1)=\frac{1}{9}\sum_{k=0}^{2}\sigma_B(k)\ .$$

Затем, двигаясь слева направо, получим:

$$S'_R(1,j+1)=S'_R(1,j)-\sigma_R(j-1)+\sigma_R(j+2)\ ,$$
$$S'_G(1,j+1)=S'_G(1,j)-\sigma_G(j-1)+\sigma_G(j+2)\ ,$$
$$S'_B(1,j+1)=S'_B(1,j)-\sigma_B(j-1)+\sigma_B(j+2)\ .$$

Таким образом, в процессе продвижения от одной окрестности $B_3(1, j)$ к следующей за ней $B_3(1, j+1)$ мы вычитаем сумму, соответствующую самому левому столбцу старой окрестности, и прибавляем сумму, соответствующую самому правому столбцу новой окрестности.

После прохождения верхнего слоя сдвигаемся вниз и вычисляем

$$S'_R(2, j), S'_G(2, j), S'_B(2, j), 1 \le j \le n-2.$$

Необходимо вычислить аналогичные суммы для столбцов высотой 3. Это можно сделать таким способом. Удаляем из столбца его верхний элемент и добавляем снизу новый. В результате получим:

$$\sigma_R(k) := \sigma_R(k) - S_R(0,k) + S_R(3,k),$$
$$\sigma_G(k) := \sigma_G(k) - S_G(0,k) + S_G(3,k),$$
$$\sigma_B(k) := \sigma_B(k) - S_B(0,k) + S_B(3,k).$$

Таким образом, имеем равенства

$$\sigma_R(k) = \sum_{i=1}^{3} S_R(i,k),\ \sigma_G(k) = \sum_{i=1}^{3} S_G(i,k),\ \sigma_B(k) = \sum_{i=1}^{3} S_B(i,k).$$

С их помощью мы можем двигаться внутри следующего слоя, состоящего из первого, второго и третьего рядов точек. Легко видеть, что этот алгоритм можно продолжать от слоя к слою, причем при каждой следующей итерации алгоритма большая часть данных используется с предыдущей итерации.

Релизация медианного фильтра

В фильтре используется сортировка со слиянием и удалением элементов. В общем случае имеется квадрат точек (здесь, как и в предыдущем разделе, описана реализация фильтра с окрестностью размером 3×3). Данные, соответствующие этим точкам, уже отсортированы при предыдущей итерации алгоритма и представлены в виде упорядоченного списка. Данные, соответствующие правому столбцу, также сортируются и представляются в виде списка. После этого эти два списка накладываются один на другой, одновременно отбрасываются данные по самому левому столбцу.

Таким образом, используя сортировку со слиянием и удалением элементов, мы получаем данные, соответствующие следующему квадрату точек, причем эти данные уже отсортированы. Выбирая элемент, расположенный

в середине списка, мы будем иметь именно тот элемент, который нам нужен для медианной фильтрации.

Обработав верхний слой, сдвигаемся вниз. Предположим, получился столбец

$$\begin{pmatrix} (i-1,k) \\ (i,k) \\ (i+1,k \end{pmatrix}.$$

Затем мы сортируем этот столбец в порядке возрастания, принимая во внимания значения $S_R(l,k), l = i-1, i, i+1$. Другими словами, мы получим отсортированный список значений компоненты цвета точек, формирующих данный столбец.

Теперь нам нужен отсортированный список, соответствующий столбцу

$$\begin{pmatrix} (i,k) \\ (i+1,k) \\ (i+2,k \end{pmatrix},$$

который получается сдвигом вниз на один пункт. Его можно получить, используя сортировку с добавлением и удалением элементов. В данном случае мы удаляем верхний элемент и добавляем недостающий нижний элемент. Затем аналогичным образом сдвигаемся слева направо, чтобы построить следующий столбец.

Приложение 2

РЕЗУЛЬТАТЫ ИСПЫТАНИЙ

Испытания программы произведены на ЭВМ со следующей системной конфигурацией: процессор Intel Pentium III (тактовая частота 500–1200 МГц), объем ОЗУ 128–256 Мб, управляющая ОС — Microsoft Windows 2000.

Применялась следующая методика тестирования. Выбирается исходное изображение, из него (или из других изображений) выделяются образцы, которые необходимо найти. Затем несколько раз выполняется поиск выде-

ленных фрагментов. Статистика времени поиска вычисляется по результатам теста на нескольких различных исходных изображениях.

Тест 1. Среднее время поиска образца. Исходное изображение имеет размер 1024 на 768 пикселов, образцы выбирались размером 64 на 64 пиксела.

Тест 2. Среднее время поиска повернутого образца. Исходное изображение имеет размер 1024 на 768 пикселов, выбран образец размером 64 на 64 пиксела, который затем размножен поворотом на углы от 0° до 350° с шагом в 10°. Поворот образца выполнен при помощи пакета Adobe Photoshop. Аналогично производился поиск для каждого из полученных изображений.

Тест 3. Среднее время поиска фрагмента, не присутствующего на исходном изображении. Исходное изображение имеет размер 1024 на 768 пикселов, выбраны 5 образцов размером 64 на 64 пиксела, не присутствующие на исходном изображении и имеющие различные цветовые характеристики, от сходных характеристик исходного изображения до, в некотором смысле, противоположных. Соответственно, проводились попытки поиска для каждого из изображений.

Ж. Л-Д. Дылыков, Ф. А. Мурзин

СИСТЕМА TRIZ_COMPUTING*

ВВЕДЕНИЕ

Успех в любой области жизни — деловой, общественной или частной — напрямую зависит от умения вести переговоры. Переговоры это факт нашей повседневной жизни. Вы обсуждаете с начальником свое повышение по службе, и даже при покупке товара в магазине вы все равно вступаете в процесс переговоров. Хотя переговоры происходят каждый день, вести их как следует нелегко. Стандартная переговорная стратегия очень часто оставляет чувство неудовлетворенности, изнурения или отчуждения, а нередко и всего вместе.

Люди оказываются перед дилеммой. Они видят лишь две возможности ведения переговоров быть податливым или жестким. Мягкий по характеру человек, желая избежать конфликта, с готовностью идет на уступки. Однако в итоге он чувствует себя ущемленным и остается в обиде. Жесткий участник, желая победить, не идет ни на какие уступки, нередко встречает такую же жесткую реакцию, которая изматывает его самого и его ресурсы, а также портит отношения с другой стороной. Есть третий путь ведения переговоров, основанный не на слабости или твердости, а скорее объединяющий и то и другое.

Конечной целью переговоров является принятие решения, устраивающее все стороны. Существует множество методов нахождения решения и его оценки. Одним из первых методом принятия решения был метод проб и ошибок, суть которого заключается в выдвижении и рассмотрении всяческих гипотез. При этом всякий раз неудачная теория отбрасывается, а вместо нее выдвигается другая. Ясно, что такая технология мало эффективна. Далее появился более “продвинутый” метод “мозгового штурма”. Мозговой штурм — психологический метод, но его автор Алекс Осборн не психолог. В основе метода — четкая мысль: процесс генерирования идей необходимо отделить от процесса их оценки. Осборн предложил вести генерирование идей в атмосфере, когда всякая критика запрещена, что позволяет высказы-

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

вать смелые, неожиданные идеи, которые обычно не решаются высказывать. Для этого отбирают небольшую и по возможности разнородную группу «генераторов идей». Высказанные идеи записываются, затем весь материал передают экспертам для оценки и отбора перспективных идей. При мозговом штурме можно в какой-то мере управлять мышлением, но суть дела от этого не меняется: поиск по-прежнему ведется простым перебором вариантов.

Также известны некоторые другие методы, такие как:

- синектика (У. Гордон);
- фокальные объекты (Ч. Вайтинг);
- гирлянды случайностей и ассоциаций (Г. Буш, СССР);
- списки контрольных вопросов (Д. Пойа, А. Осборн, Т. Эйлоарт) и т.д.

1. СИСТЕМА ТРИЗ

В 1965 году Генрих Саулович Альтшуллер предложил теорию решения изобретательских задач (ТРИЗ). Основным инструментом ТРИЗ является алгоритм решения изобретательских задач (АРИЗ). АРИЗ — комплексная программа алгоритмического типа, основанная на законах развития технических систем и предназначенная для анализа и решения изобретательских задач. Однако сейчас встал вопрос о переносе инструмента ТРИЗ на другие области деятельности человека, такие как реклама, бизнес и т.д. Еще одной интересной областью ее применения является психология, вернее та ее часть, которая занимается устранением конфликтов и ведением переговоров.

Опишем теорию решения изобретательских задач. Основным инструментом в ТРИЗ является алгоритм решения изобретательских задач (рис. 1).

Алгоритм состоит из следующих этапов.

1. Анализ задачи:

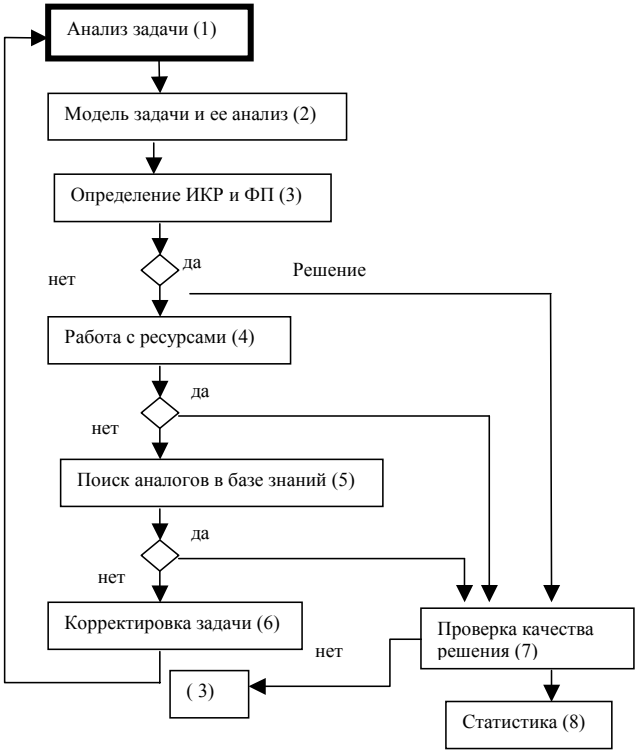
а) записать условие мини-задачи (без специальных терминов) “Техническая система для ... включает ... Техническое противоречие №1: ... Техническое противоречие №2: ... Необходимо при минимальных изменениях в системе ... ” так записывается модель задачи;

б) записать конфликтующую пару (изделие и инструмент);

в) составить диаграмму полезности.

2. Анализ модели задачи:

а) определить оперативную зону (ОЗ)[1];
б) определить оперативное время (ОВ)[2];
в) определить вещественно-полевые ресурсы (ВПР)[3]
- внутрисистемные (ВПР инструмента, ВПР изделия),
- внешнесистемные (ВПР среды),
- надсистемные (отходы постоянной системы, «копеечные»).



[1] Оперативная зона — это пространство, в пределах которого возникает конфликт, указанный в модели задачи.

[2] Оперативное время — это имеющиеся ресурсы времени: конфликтное время T1 и время до конфликта T2.

[3] Вещественно-полевые ресурсы — это вещества и поля, которые уже имеются или могут быть легко получены.

3. Определение идеального конечного результата и физического противоречия:

- ИКР-1: Икс элемент, не усложняя систему и не вызывая вредных явлений, устраняет ... в течение ОВ в пределах ОЗ.
- ФП — противоречивые требования к физическому состоянию ОЗ.

4. Применение информфонда.

5. Анализ способа устранения ФП.

6. Изменение или замена задачи.

7. Анализ хода решения.

8. Применение полученного решения.

По аналогии с АРИЗ введем алгоритм решения проблемных ситуаций (АРПС).

В соответствии с АРПС, анализ начинается с формулировки основной функции системы, ее состава и нежелательного эффекта (НЭ), возникающего в процессе функционирования системы.

Итак, система «Переговоры» для удовлетворения одной или нескольких потребностей состоит из участников общения А и В, цели общения и межличностных отношений. В процессе общения возникает нежелательный эффект НЭ1 — претензия участника общения А к участнику В. НЭ1 чаще всего возникает, когда один участник общения в процессе какой-либо деятельности предъявляет к другому участнику новые повышенные требования или пытается что-то изменить в сложившейся системе общения, чтобы она выполняла для него дополнительные функции.

Для удовлетворения этой претензии участник В предлагает средство устранения (СУ) — некоторое действие, приводящее к устранению НЭ1. Если предлагаемое СУ удовлетворяет участника А, то нежелательный эффект НЭ1 устраняется и конфликт не возникает. Если же предлагаемое СУ не удовлетворяет участника А, то в ситуации возникает новый нежелательный эффект НЭ2.

Отношения между НЭ1, СУ и НЭ2 связаны причинно-следственной связью:

1) если ввести СУ, то НЭ1 устраняется, но возникает НЭ2;

2) если же СУ не вводить, то НЭ2 не возникает, но сохраняется НЭ1.

Такая форма причинно-следственной связи создает противоречие — т. е. такое свойство связи между двумя взаимодействующими участниками общения, при котором нужное для участника А изменение формы общения вызывает недопустимое изменение формы общения для участника В, и наоборот.

Постановка задачи по предотвращению конфликта в идеальном варианте может быть сформулирована следующим образом: не вводя СУ и тем самым не создавая НЭ2, устранить НЭ1.

Определим оперативную зону, в которой сталкиваются несогласуемые (в общем случае — противоположные) интересы участников общения. Так как основной функцией общения является удовлетворение потребностей, то очевидно, что оперативной зоной является отношение каждого из участников общения к содержанию потребностей и форме их удовлетворения.

Для поиска эффективного решения также важно принимать во внимание оперативное время — период возникновения претензии и протекания самого конфликта.

Сам конфликт в АРПС выступает как физическое противоречие для данной проблемы и может быть сформулирован как предъявление каждым участником в ходе общения противоположных требований к одному и тому же:

- объекту потребности (например: «Эта земля должна быть моей, а не его, чтобы моя столица была дальше от границы!»);
- содержанию его потребности (например: «Я сделаю то, что я хочу, а не то, что ты хочешь, потому что я лучше тебя знаю, что правильно!»);
- способу реализации потребности (например: «Отдыхать мы будем в горах, а не на море, потому что я люблю кататься на лыжах!»);

Идеальный конечный результат (ИКР): необходимо найти в содержании потребностей каждого участника такие ресурсы, которые позволят получить результат общения, удовлетворяющий одного из участников и не вызывающий негативного состояния другого участника. Специалисты по ведению переговоров определяют его так: ”Не ведите позиционный торг! Говорите об интересах, а не о позициях”.

2. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

2.1. Информационный фонд

АРИЗ предполагает использование информационного фонда. В него входят:

- приемы устранения технических и физических противоречий;

- банк закономерностей и тенденций развития объектов техники;
- банк методов инженерного творчества ;
- банк методов активизации инженерного творчества;
- банк открытий и физических эффектов (выбор принципа функционирования технического решения);
- банк отрицательных эффектов;
- банк приемов преобразования объектов техники;
- банк методов оценки технико-экономических параметров объектов, их работоспособности и эффективности (выявление базового объекта лучшего мирового образца).

Информфонд строится на принципах классификации и систематизации. Особую актуальность получил банк отрицательных эффектов, так как в последние десятилетия обнаружилось много "отдаленных" последствий загрязнения окружающей среды и информационной составляющей, что создает потенциальную угрозу существованию самой жизни на Земле.

Информфонд в АРПС содержит набор типичных примеров решения конфликтных ситуаций. При решении своей конфликтной проблемы просматривается информфонд на наличие подобных примеров.

2.2. СТРУКТУРА ПРОГРАММЫ

Для создания программы моделирования методов принятия решений в системе ТРИЗ на примере загадок была выбрана система Microsoft Visual C++, благодаря следующим ее особенностям:

- легкости прототипирования и создания программ с помощью интеллектуальных шаблонов (wizards) на базе библиотеки классов MFC;
- наличию библиотеки стандартных шаблонов STL и простоте ее использования;
- наличию объектно-ориентированной среды, повышающей скорость и качество разработки программ.

Программа имеет модульную структуру. Общая схема программы показана на рис. 1.

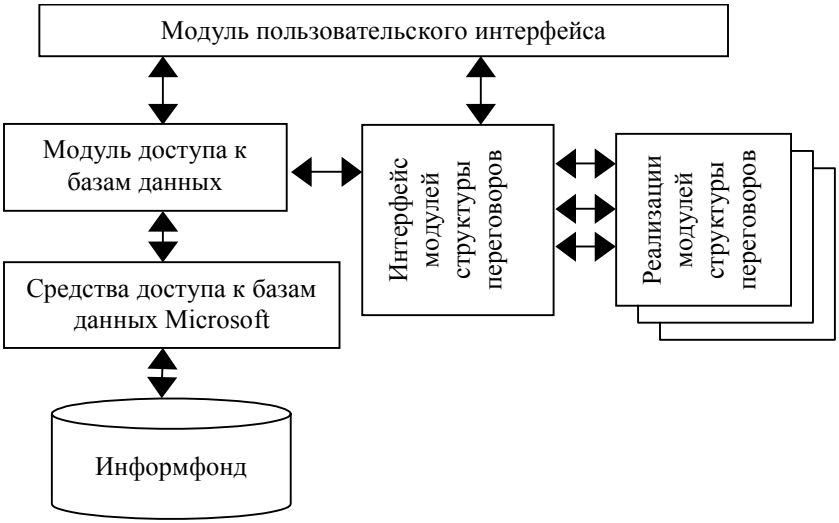


Рис. 1. Структура программы

Модуль пользовательского интерфейса создан на базе библиотеки классов MFC (рис. 2) и представляет собой MDI (Multiple Document Interface), разделенный на две части.

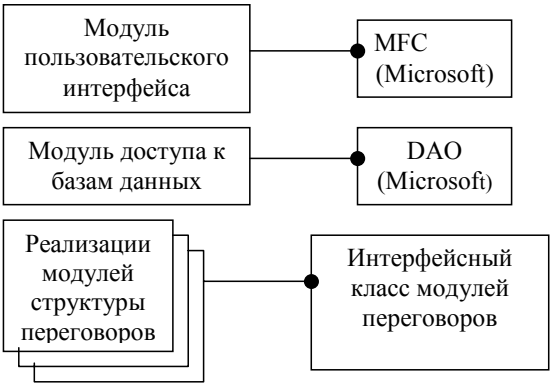


Рис.2. Иерархия наследования классов программы

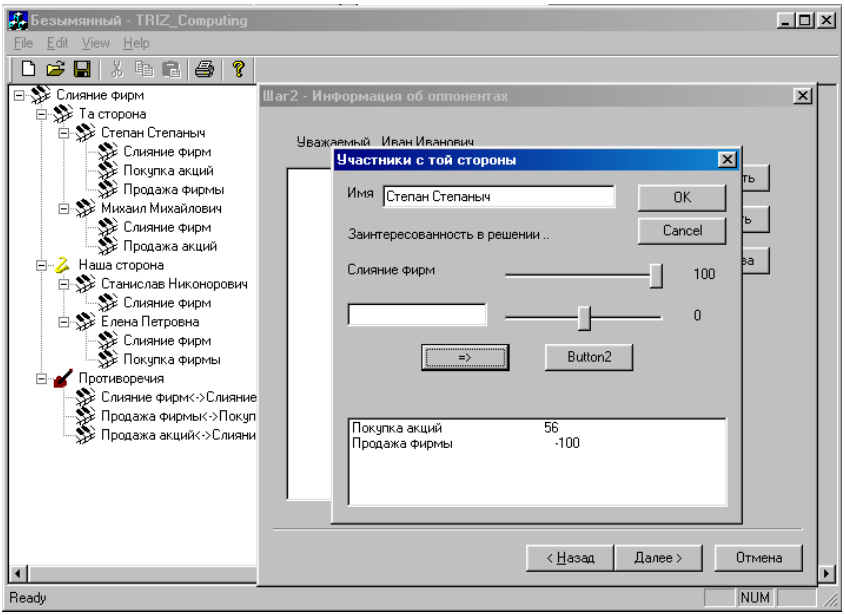


Рис. 3. Система TRIZ_Computing

В левой части отображается структура переговоров в виде дерева, а в правой — параметры. В его функции входит обеспечение связи с пользователем: выбор конкретного класса моделируемых переговоров и возможность задания параметров переговоров. Модуль пользовательского интерфейса позволяет также выбрать базу данных для использования ее в качестве информационного фонда. Также предусмотрен режим Wizard'a для заполнения структуры.

Модуль доступа к базам данных адаптирует данную программу к стандартным средствам доступа фирмы Microsoft. Использование стандартных средств системы делает её независимой от конкретного формата базы данных. Более того, в таком варианте возможны моделирование структуры базы данных и её наполнение с помощью специально созданных для этого средств. В данной работе моделирование и построение базы данных тестового информационного фонда было проведено с помощью СУБД Microsoft Access из широко распространённого пакета Microsoft Office.

3. СТРУКТУРА ДАННЫХ

Для анализа конфликтных ситуаций предложена структурная схема, изображенная на рис. 4.

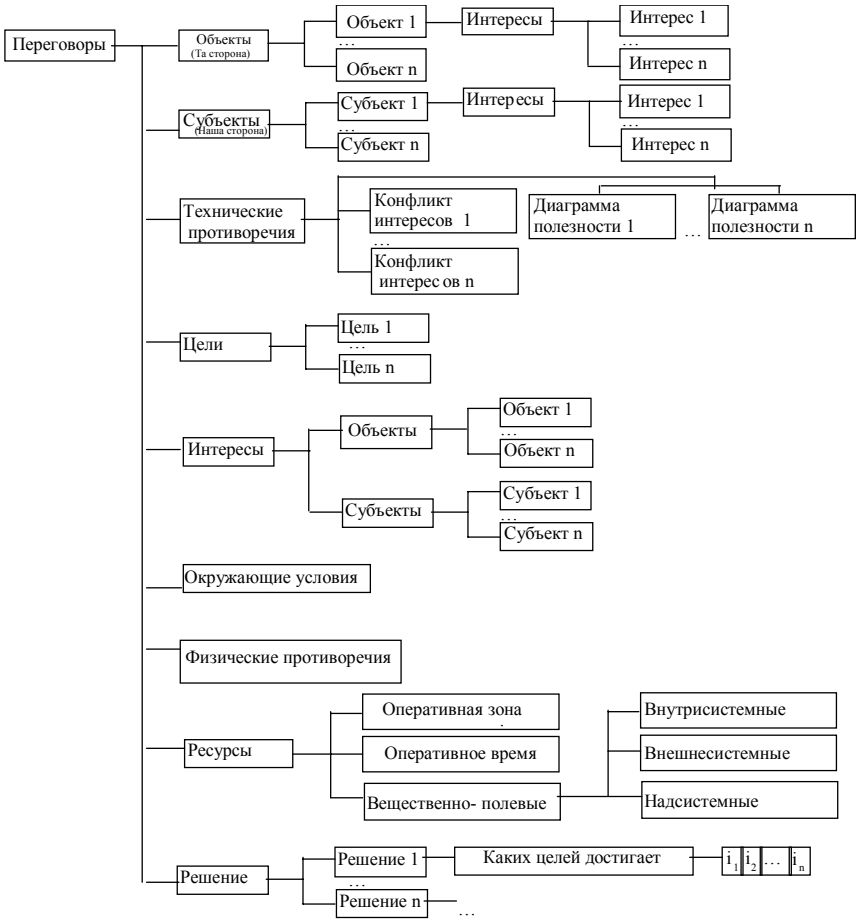


Рис. 4. Структура данных

4. ЗАКЛЮЧЕНИЕ

В данной работе были рассмотрены методы ТРИЗ и их применение к проблеме разрешения конфликтов. В ходе работы был рассмотрен алгоритм АРИЗ и его аналог для решения конфликтных ситуаций АРПС, также автором была изучена теория экспертных систем.

В результате стало возможным написание графической оболочки TRIZ_Computing для разбора конфликта. Для этих целей была использована среда программирования VisualC++ 6.0. Она реализована как экспертная система.

СПИСОК ЛИТЕРАТУРЫ

1. **Альтшуллер Г. С.** Найти идею. — Новосибирск.: Наука, 1991.
2. **Правила** игры без правил / Сост. Селюцкий А. Б. — Петрозаводск: Карелия, 1989.
3. **Матвеев Л. А.** Компьютерная поддержка решений. — СПб: Специальная Литература, 1998.
4. **Макаров И. М., Виноградская Т. М. и др.** Теория выбора и принятия решения. — М.: Наука, 1982.
5. **Саати Т.** Принятие решений. — М.: Радио и связь, 1993.
6. **Фишер Р., Юри У.** Путь к согласию. — М.: Наука, 1990.
7. **Шрагина Л. И.** Анатомия конфликта // Журнал практического психолога. — Москва, 1999.
8. **Ниренберг Дж.** Маэстро переговоров. — Минск: Парадокс, 1997.
9. **Элти Дж.** Экспертные системы: концепции и примеры. — М.: Финансы и статистика, 1987.
10. **Сойер Б., Фостер Д. Л.** Программирование экспертных систем на Паскале. — М.: Финансы и статистика, 1990.
11. **Уотерман Д.** Руководство по экспертным системам. — М.: Мир , 1989.

Ж.Л-Д.Дылыков, В.А.Пустыльников

АВТОМАТИЗАЦИЯ МЕТОДОВ ПРИНЯТИЯ РЕШЕНИЙ НА ЖЕЛЕЗНОДОРОЖНОМ ТРАНСПОРТЕ КАК ГАРАНТИЯ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ ДВИЖЕНИЯ*

ВВЕДЕНИЕ

Поддержка принятия решения на базе использования информационных компьютерных систем управления (ИС) предназначена для обеспечения работников различного рода данными, информацией и знаниями, облегчающими принятие ими эффективных решений. Обычно процесс принятия решения включает в себя следующие составляющие: планирование, генерирование ряда альтернатив, установления приоритетов, выбор наилучшей линии поведения после нахождения ряда альтернатив, распределение ресурсов, определение потребностей, предсказание исходов, построение систем, измерение характеристик, обеспечение устойчивости системы, оптимизация и разрешение конфликтов. Рассмотрим пример автоматизации процесса принятия решения на примере работы цеха эксплуатации локомотивного депо железнодорожного транспорта.

Добиться эффективной и высокопроизводительной работы локомотивного хозяйства без оперативной информационной поддержки невозможно. Центральными вопросами в организации работы железнодорожного транспорта всегда были и будут вопросы обеспечения безопасности движения поездов. Ключевым звеном в вопросах обеспечения безопасности эксплутационной работы является локомотивная бригада.

1. ТРЕБОВАНИЯ К РАБОТЕ ЦЕХА ЭКСПЛУАТАЦИИ

В последнее десятилетие наряду с активным развитием технических средств, предназначенных для безусловного обеспечения безопасности движения поездов, произошли значительные изменения в отношении к локомотивной бригаде как основному элементу в технологии перевозочного

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

процесса. На уровне МПС РФ разработаны и утверждены единые правила формирования и использования локомотивных бригад; определены условия допуска машинистов к работе в одно лицо (только машинист, без помощника на специально оборудованном локомотиве), изложенные в нормативных документах [4, 5].

На современном этапе развития железных дорог при переходе на длинные плечи (работа локомотива без смены до 12 часов) обслуживания работа с локомотивной бригадой рассматривается как непрерывный технологический процесс, в котором можно выделить следующие составляющие:

- процедура формирования локомотивной бригады;
- обеспечение режима труда и отдыха бригады в соответствии с установленными нормативами;
- постоянный мониторинг психологического и физического состояния членов бригады;
- контроль и анализ эксплуатационной работы бригады с помощью технических средств (скоростемерных лент);
- работа над повышением уровня технической грамотности членов бригады;
- комплексные мероприятия по физической и психологической реабилитации членов бригады в случае необходимости (на основании данных мониторинга);
- допуск бригады (машиниста в одно лицо) к работе только в случае безусловного выполнения всех требований нормативной документации.

За соблюдение всех правил и требований, предъявляемых к формированию и использованию локомотивных бригад, отвечает персонал цеха эксплуатации: дежурные по депо, нарядчики, старший нарядчик, инженер-психолог, машинисты-инструкторы и другие работники цеха эксплуатации, ответственные за использование локомотивных бригад. Они должны своевременно обеспечивать друг друга необходимой для принятия решений информацией, и сами оперативно принимать решения, учитывая, анализируя и сопоставляя не только входящую информацию о работнике, но и всю документацию, регламентирующую формирование и использование бригад, — правила, инструкции, дополнения к инструкциям и т.д. Как показывает опыт, при больших объёмах эксплуатационной работы в депо, где контингент машинистов и помощников исчисляется сотнями, ошибки в формировании и использовании локомотивных бригад практически неизбежны. Более того, зачастую это не просто случайные ошибки, а сознательные нарушения, обоснованные жизненной необходимостью.

Комплексно и системно решить эти проблемы, исключив человеческий фактор при решении ключевых вопросов обеспечения безопасности эксплутационной работы — вопросов формирования и использования локомотивных бригад, можно только путём комплексной автоматизации работы цеха эксплуатации.

При разработке таких информационно-управляющих систем необходимо закладывать определенные базовые принципы их функционирования. Не претендуя на полноту, отметим наиболее значимые требования: высокая надежность, обеспечивающая сохранность и целостность информации; высокая производительность, поддерживающая работу системы в реальном времени; удобный пользовательский дизайн программ; гибкость, обеспечивающая стыковку с другими системами; наращиваемость, предполагающая определенную открытость системы; тесная интеграция баз данных в Web.

Современный рынок программ предлагает широкий выбор различных универсальных и специализированных сред программирования для создания информационно-управляющих систем. Целесообразность использования того или иного программного продукта зависит в первую очередь от задач, структуры локальных сетей, количества компьютеров и ожидаемой нагрузки на предприятии. Локомотивные депо можно отнести к предприятиям средней группы. Для них оптимальной с точки зрения соотношения качества и цены является сетевая технология MS SQL Server. MS SQL Server 2000 является достаточно мощной современной системой управления базами данных, позволяющей в том числе использовать механизм репликаций для синхронизации данных в структурных элементах базы.

Лабораторией автоматизации управления локомотивным хозяйством разрабатывается сетевой комплекс АРМ цеха эксплуатации. Данный комплекс не только обеспечивает решение поставленных вопросов, но и позволяет выстроить стройную технологическую цепочку по работе с локомотивными бригадами в соответствии с требованиями сегодняшнего дня. В ряде локомотивных депо Западно-Сибирской железной дороги уже успешно внедрены первые составляющие комплексной системы управления цехом эксплуатации — АРМ “Нарядчика” и АРМ “Зав.бригадами”, полностью исключающей ошибки оперативного персонала в формировании бригад и планировании их работы.

2. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

2.1. Требования к аппаратному и программному обеспечению

Сервер

Требования к аппаратному и программному обеспечению сервера определяются требованиями используемой версии SQL Server 2000 (рекомендуется Windows NT4/2000 Server). Рекомендуется не менее PIII-500 256 RAM, HDD 20 GB.

Рабочие места (клиентская часть)

На рабочих местах должен быть установлен Windows 98 SE2. Рекомендуется использование Windows NT 4.0 Workstation или Windows 2000 Professional. В качестве рабочих мест необходимо использовать компьютеры не ниже P166MMX 32 RAM.

По общей схеме организации системы Клиент-сервер программа клиента размещается на рабочей станции (см. рис.1).

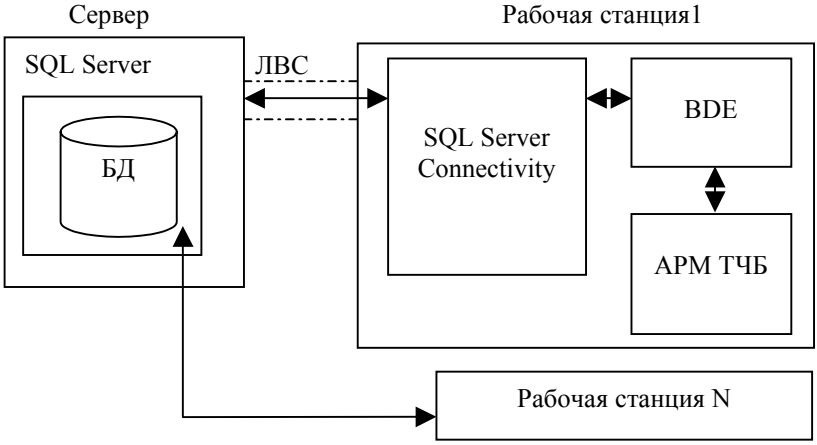


Рис. 1. Общая схема организации системы клиент-сервер

Программный комплекс АРМ ТЧБ состоит из серверной части, которая осуществляет хранение информации и организует доступ к ней, и клиентской, которая осуществляет вывод данных в удобном для пользователя виде, а также организует ввод и редактирование данных.

Серверная часть комплекса представляет собой СУБД SQL Server 2000, к которому осуществляется доступ по локальной сети с рабочих мест, на которых установлено клиентское программное обеспечение.

2.2. Описание программы

Интерфейс программы представлен на рис. 2.

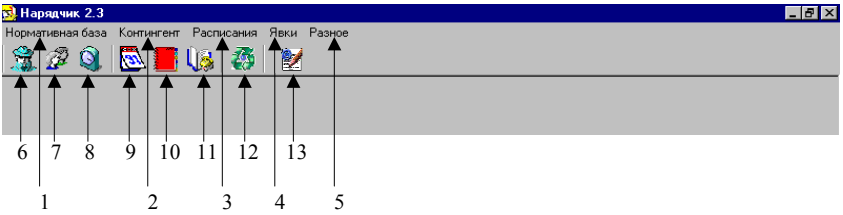
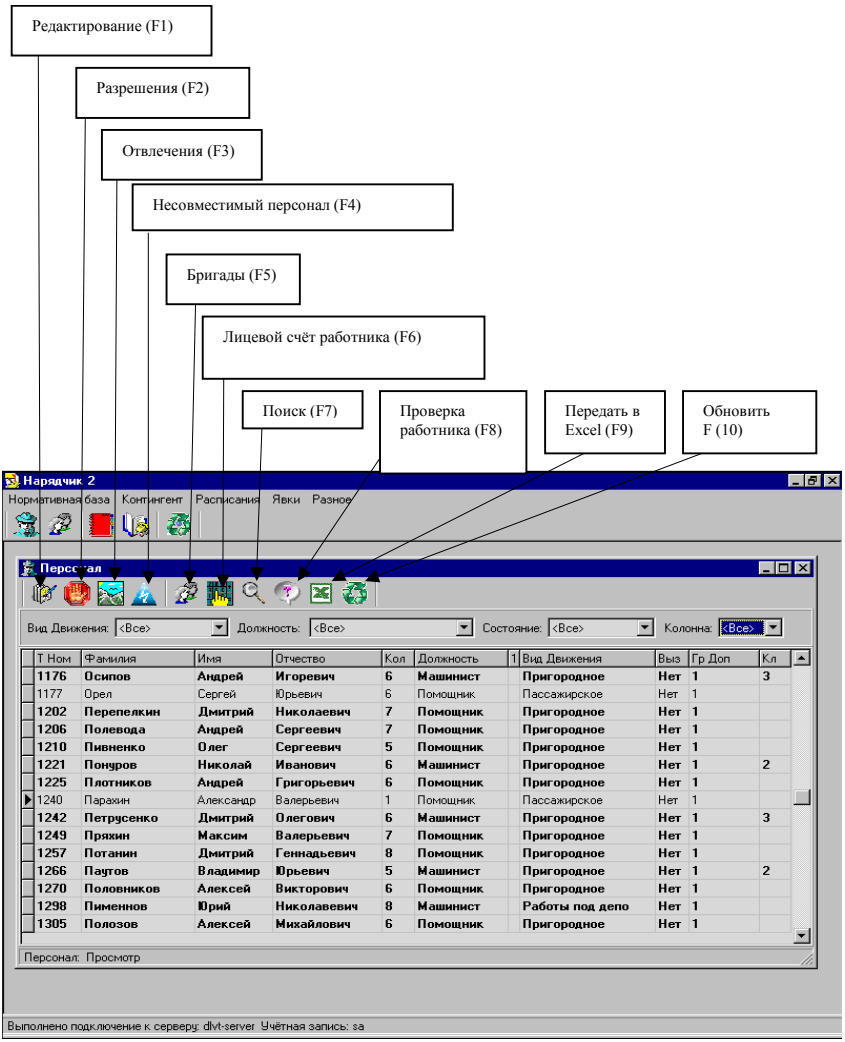


Рис. 2. Интерфейс программы

1. Пункт меню «Нормативная база» содержит вызов форм:
   - календарь,
   - места работ.
2. Пункт меню «Контингент» включает:
   - персонал,
   - бригады,
   - переработка.
3. Пункт меню «Расписания» содержит информацию по видам расписаний:
   - расписание пассажирских поездов,
   - ежедневно:е расписание.
4. Пункт «Явки» вызывает форму
   - журнал явок.
5. Пункт «Разное» содержит вспомогательные функции:
   - обновить всё,
   - о программе.

Нормативная база обязательный составной элемент данной программы и представляет собой две экранных формы, выполненные в виде активно-информационных окон. Открыть данные окна возможно, выбрав соответствующий названию окна подпункт в пункте меню «Нормативная база».

2.2.1. Описание информационного окна «Персонал»



Т.Ном	Фамилия	Имя	Отчество	Кол	Должность	1	Вид Движения	Выз	Гр Доп	Кл
1176	**Осипов**	**Андрей**	**Игоревич**	**6**	**Машинист**		**Пригородное**	**Нет**	**1**	**3**
1177	Орел	Сергей	Юрьевич	6	Помощник		Пассажирское	Нет	1	
1202	**Перепелкин**	**Дмитрий**	**Николаевич**	**7**	**Помощник**		**Пригородное**	**Нет**	**1**	
1206	**Полевода**	**Андрей**	**Сергеевич**	**7**	**Помощник**		**Пригородное**	**Нет**	**1**	
1210	**Пивненко**	**Олег**	**Сергеевич**	**5**	**Помощник**		**Пригородное**	**Нет**	**1**	
1221	**Понуров**	**Николай**	**Иванович**	**6**	**Машинист**		**Пригородное**	**Нет**	**1**	**2**
1225	**Плотников**	**Андрей**	**Григорьевич**	**6**	**Помощник**		**Пригородное**	**Нет**	**1**	
1240	Парахин	Александр	Валерьевич	1	Помощник		Пассажирское	Нет	1	
1242	**Петрусенко**	**Дмитрий**	**Олегович**	**6**	**Машинист**		**Пригородное**	**Нет**	**1**	**3**
1249	**Пряхин**	**Максим**	**Валерьевич**	**7**	**Помощник**		**Пригородное**	**Нет**	**1**	
1257	**Потанин**	**Дмитрий**	**Геннадьевич**	**8**	**Помощник**		**Пригородное**	**Нет**	**1**	
1266	**Паутов**	**Владимир**	**Юрьевич**	**5**	**Машинист**		**Пригородное**	**Нет**	**1**	**2**
1270	**Половников**	**Алексей**	**Викторович**	**6**	**Помощник**		**Пригородное**	**Нет**	**1**	
1298	**Пименнов**	**Юрий**	**Николаевич**	**8**	**Машинист**		**Работы под депо**	**Нет**	**1**	
1305	**Полозов**	**Алексей**	**Михайлович**	**6**	**Помощник**		**Пригородное**	**Нет**	**1**	

Рис. 3. Основное окно программы

Открыть данное окно можно двумя способами:

1) выбрав соответствующий подпункт в пункте меню «Контингент» (2.2),
2) кнопкой «Персонал» (6).

Основная информация о каждом работнике представлена в виде таблицы (рис.3) и содержит 11 основных параметров, необходимых нарядчику для оперативной работы.

1. Табельный номер.
2. Фамилия.
3. Имя.
4. Отчество.
5. Номер колонны.
6. Должность.
7. Возможность работать в одно лицо (если +, то может).
8. Вид движения.
9. Возможность вызова работника (да/нет).
10. Группа профпригодности.
11. Класс квалификации (для машинистов).

В этом окне указываются все параметры персонала, с учетом которых далее будет приниматься решение об объединении в бригады и отправке бригады в рейс.

Не имеет смысла приводить здесь полное описание программы. Отметим, что данная программа с успехом прошла опытную эксплуатацию в локомотивных депо Западно-Сибирской железной дороги и теперь распространяется по всей сети железных дорог МПС РФ.

2.3. Установка клиентской части комплекса

На рабочем месте должны быть установлены следующие компоненты:

- SQL Server Connectivity – устанавливается из дистрибутива SQL Server 2000;
- BDE for SQL Server поставляется вместе с программой. Также BDE можно установить из дистрибутива Borland Delphi 5 или Borland C++ Builder 5.

После установки перечисленных компонентов необходимо настроить соединение с сервером и установить настройки для отправления сообщений в систему оперативного контроля за дислокацией бригады (ОКДБ).

Имя пользователя и пароль: вводятся в окне соединения, которое появляется при запуске программы.

Имя компьютера сервера, имя сервера БД, имя БД, ключ, папка исходящих сообщений в систему оперативного контроля за дислокацией локомотивных бригад (ОКДБ), программа отправки сообщений ОКДБ: вводятся в форме, которая появляется при нажатии на кнопку «Настройка» в окне соединения. Ключ представляет собой уникальное числовое значение, которое передаётся разработчиками при установке программы (рис. 4).

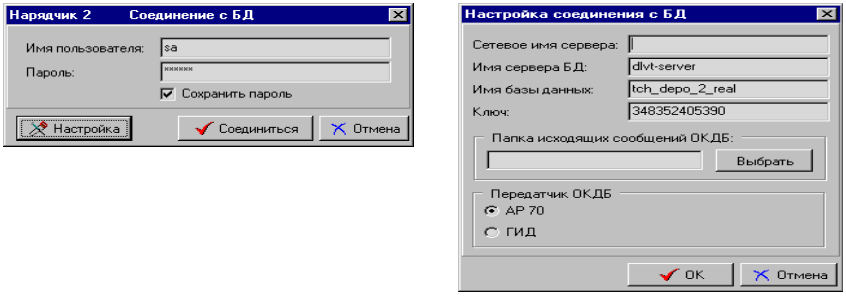


Рис. 4. Окна настройки соединения с БД

Программа позволяет также выполнять запуск из командной строки со следующими параметрами:

NoDialog – не показывать диалог настройки соединения.

Если диалог не отображается, то при соединении используются последние настройки программы или настройки, указанные в параметрах соединения с сервером БД и настройках ОКДБ. Параметры имеют следующий формат:

<имя_параметра>:<значение>
Host – сетевое имя сервера; **Server** – имя сервера БД; **Database** – имя БД;
User – имя пользователя; **Password** – пароль; **Key** – ключ;
OKDBOut – исходящая папка сообщений ОКДБ;
Prog – передатчик сообщений ОКДБ (0 – АП70, 1 – ГИД).

Имена параметров регистронезависимы. Ниже представлен пример запуска программы:

D:\CTCH\TCHB\TCHB.exe nodialog host: server:Altayskaya
database:tch_depo_2_real user:sa password:egor okdbout:c:\asoup\out prog:1

Если параметры не указаны, то программа запускается обычным образом.

ЗАКЛЮЧЕНИЕ

Находящиеся в эксплуатации информационные системы базируются на системе АСОУП (автоматическая система оперативного управления поездами) и собирают первичную информацию о дислокации локомотивов и локомотивных бригад из различных источников. В локомотивном хозяйстве первичными звеньями в этих цепочках являются линейные предприятия: основные и оборотные локомотивные депо, а также дома отдыха локомотивных бригад. Для получения надежной и достоверной информации с низовых звеньев необходимо автоматизировать рабочие места, перенеся центр тяжести сбора, обработки и первичного анализа информации на специализированные программы.

СПИСОК ЛИТЕРАТУРЫ

1. **Матвеев Л. А.** Компьютерная поддержка решений. — СПб: Специальная литература, 1998.
2. **Макаров И. М., Виноградская Т. М. и др.** Теория выбора и принятия решения. — М.: Наука,1982.
3. **Саати Т.** Принятие решений. — М.: Радио и связь, 1993 .
4. **Приказ** 8ЦЗ от 18.09.1990 г. “О введении в действие особенностей регулирования рабочего времени и времени отдыха отдельных категорий работников ж.д. транспорта”.
5. **Приложения** к приказу МПС от 16.06.1994 г. № 1ЦЗ “Основные мероприятия планово-предупредительной системы обеспечения безопасности движения в локомотивном хозяйстве”.

В. А. Евстигнеев

КОНВЕЙЕРНАЯ МОДЕЛЬ ПЕРЕВОЗОК КАК МОДЕЛЬ ПЕРЕСЫЛКИ ПРОТЯЖЕННЫХ СООБЩЕНИЙ*

1. ВВЕДЕНИЕ

Хорошо известная транспортная задача по стоимости сводится к отысканию наибольшего потока в двухполюсной сети с наименьшей стоимостью и не содержит в явном виде понятия времени. Введение времени в модель перевозки однородного груза требует пересмотра основных условий задачи. Так вместо указания общего количества груза, перевозимого по данной дуге, как это делается в обычной сетевой транспортной задаче, указывается наибольшее количество груза, перевозимого за единицу времени, причем поток в новом смысле удовлетворяет всем ограничениям в определении потока в старом смысле. Но изменяется модель перевозок: груз считается разбитым на порции, каждая из которых есть наибольшее количество груза, перевозимого за единицу времени по данной дуге. Поэтому общее количество перевозимого по дуге груза определяется, кроме всего прочего, временем, в течение которого осуществляются перевозки. Подобный характер организации перевозок называется *конвейерной моделью перевозок*.

Интерес к этой задаче, которой автор занимался в конце 60-х–начале 70-х гг. [1–3], может пробудить так называемая "червеобразная" модель пересылки сообщений в мультипроцессорных и мультикомпьютерных системах. Эта модель предполагает, что сообщение занимает в сети межпроцессорной связи заметный временной интервал, что позволяет сравнивать сообщение с "червем" с фиксированными длиной и шириной. Это эквивалентно предположению о том, что сообщение представимо в виде "цуга" (последовательности) порций информации, величина которых равна ширине "червя", а их количество равно длине "червя".

В простейшем случае, когда ширина сообщения не изменяется при движении по различным дугам, задача не отличается оригинальностью и сводится к задаче о нахождении кратчайшего пути в сети, получаемой из данной удалением дуг, пропускная способность которых меньше ши-

*Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

рины “червя”. Другое дело, если ширина сообщения зависит от такого параметра, как пропускная способность дуги. При этом ширина сообщения, пересылаемого по пути P, состоящего из k дуг, равна “объему” сообщения, деленному на наименьшую пропускную способность среди k дуг, составляющих путь P. Тогда возникает минимаксная задача отыскания кратчайшего пути с наибольшей пропускной способностью.

В общем случае сообщение рассыпается на порции информации, которые пересылаются параллельно по нескольким путям различной длины и пропускной способности. Мы оставляем пока в стороне механизм рассыпания на порции и последующей сборки сообщения и рассмотрим здесь основные результаты, касающиеся конвейерной организации перевозок. Это требует уточнения ряда определений, таких как динамический поток, план перевозок, многополюсные потоки и др., которые не объяснялись так подробно, как того они заслуживали, что делало чтение соответствующих статей крайне затруднительным.

Все неопределяемые здесь понятия могут быть найдены в [4].

2. СЕТЕВАЯ ЗАДАЧА ПО ВРЕМЕНИ ДЛЯ ДВУХПОЛЮСНЫХ СЕТЕЙ И ПОНЯТИЕ ПОТОКА

Указанная в заголовке задача возникает при решении задачи о наибольшем количестве груза, которое можно перевезти по сети с входом s, в котором хранится груз P, в выход q за фиксированное время T. Для ее решения напомним постановку обычной транспортной задачи по стоимостному критерию.

Пусть дана ориентированная двухполюсная сеть $G(s, q) = (V, E; s, q)$ с входом s и выходом q, каждой дуге которой сопоставлено натуральное число $c(e) > 0$, называемое *пропускной способностью дуги* e. Заметим, что случай перевозок однородного груза по многополюсной сети с входами $s_1, \ldots, s_k$ и выходами $q_1, \ldots, q_l$ легко сводится к двухполюсному случаю введением фиктивных входа s_0 и выхода q_0. Величина $c(e)$, $e \in E$, ограничивает объем перевозимого по дуге e груза. Требуется обеспечить перевозку P единиц размещенного (или производимого в s) груза по сети G, не допуская превышения пропускных способностей дуг. Математической моделью задачи о перевозках служит наибольший поток [5].

Определение 1. *Потоком* f по сети G называется целочисленная функция, определенная на множестве дуг E и удовлетворяющая следу-

ющим условиям:

1. $0 \le f(e) \le c(e)$;
2. $\sum_v f(v,w) = \sum_x f(w,x),\ w \ne s,q$.

Здесь первое суммирование ведется по дугам, заходящим в w, а второе — по дугам, исходящим из w.

Определение 2. Число $F_f = \sum_v f(s,v) = \sum_v f(v,t)$ называется *величиной (мощностью) потока* f. Поток f_0, величина F_0 которого наибольшая из всех возможных потоков по сети G, называется *наибольшим потоком*.

Очевидно, что если $P = F_0$, то построение наибольшего потока полностью решает задачу; при $P > F_0$ задача не имеет решения, при $P < F_0$ решение сводится к построению ограниченного по величине потока.

Если дугам e сети G сопоставлены числа $b(e)$ — стоимости перевозки единицы груза по дуге e, то величина $B(f) = \sum_e f(e)b(e)$ называется *стоимостью потока* f. Наибольший поток с минимальной стоимостью называется *оптимальным потоком*. Поток произвольной величины F_f, где $0 < F_f \le F_0$, с наименьшей стоимостью называется *оптимальным потоком величины* F_f.

Известно, что от одного потока f_1 к другому f_2 той же мощности можно перейти с помощью операции *сдвига на цикле*. Однако более эффективным является последовательное построение оптимального потока с помощью отыскания увеличивающей поток цепи наименьшей длины. Отыскание цепи проводится алгоритмом Дейкстры, при этом считается, что длина (стоимость, время перевозки) дуги с ненулевым потоком, проходимой против ориентации, отрицательна. Достоинство этого подхода состоит в том, что мы на каждом шаге имеем *цепное разложение потока*. Другими словами, на каждом шаге алгоритма построения потока мы имеем множество s,q-путей (хотя речь идет о цепном разложении) $\{P_1, \ldots, P_r\}$ таких, что для каждой дуги e имеем

$$f(e) = \sum_{k \mid P_k \ni e} f_k(e),$$

где суммирование ведется по всем путям P_k, содержащим дугу e.

3. ПРОСТРАНСТВЕННО-ВРЕМЕННАЯ ДИАГРАММА

Рассмотрим задачу о перевозке наибольшего количества груза по сети за заданное время. При этом, как мы уже указывали, пропускная

способность дуги $c(e)$ ограничивает количество груза, поступающего на вход дуги e за единицу времени.

Будем обозначать через $t(e)$ время перевозки единицы груза по дуге e, так что единица груза, поступив на вход дуги в момент t_0, покидает ее в момент $t_0 + t(e)$.

Утверждение 1. Пусть G представляет собой путь вида $\pi = (s = v_0, v_1, \ldots, v_k = q)$. Тогда единица груза проходит этот путь за время $t(\pi) = \sum_{i=1}^{k} t(v_{i-1}, v_i)$. За заданное время T по сети G будет перевезено

$$P(T) = \begin{cases} 0, \text{ если } T < t(\pi); \\ F(\pi), \text{ если } T = t(\pi); \\ F(\pi)(T - t(\pi) + 1), \text{ если } T > t(\pi). \end{cases}$$

Здесь

$$F(\pi) = \min_i c(v_{i-1}, v_i), \ 1 \le i \le k.$$

Заметим, что $F(\pi)$ есть величина наибольшего потока $f(\pi)$ по сети G и $\{\pi\}$ есть цепное разложение потока $f(\pi)$.

Пусть дана целочисленная решетка $L(t, v)$, строки которой соответствуют некоторой дискретной шкале времени, а столбцы — вершинам сети G, так что первый столбец соответствует входу s сети, а последний — выходу q. Элемент d_{tv} отражает состояние вершины v в момент времени t, т.е. количество груза, находящегося в данный момент в этой вершине. Итак, мы заменяем каждую вершину v множеством вершин $\{v_{t_0}, v_{t_0+1}, \ldots, v_{t_0+k}, \ldots\}$, где момент t_0 для каждой вершины определяется как наименьшее время, требуемое для достижения вершины v из входа s. Эти вершины соединим дугами $(v(t), v(t+1))$. Каждую вершину v_{it} соединяем дугой с вершиной $v_{(j,t+t(i,j))}$ для всех пар (v_i, v_j), соответствующих дугам в исходной сети, и для всех моментов времени, больших начального момента t_0, определяемого для каждой вершины.

Граф, построенный на целочисленной решетке $L(t, v)$, называется **пространственно-временной диаграммой** (ПВД).

Если сеть G состоит из одного пути π, то пространственно-временная диаграмма будет иметь вид:

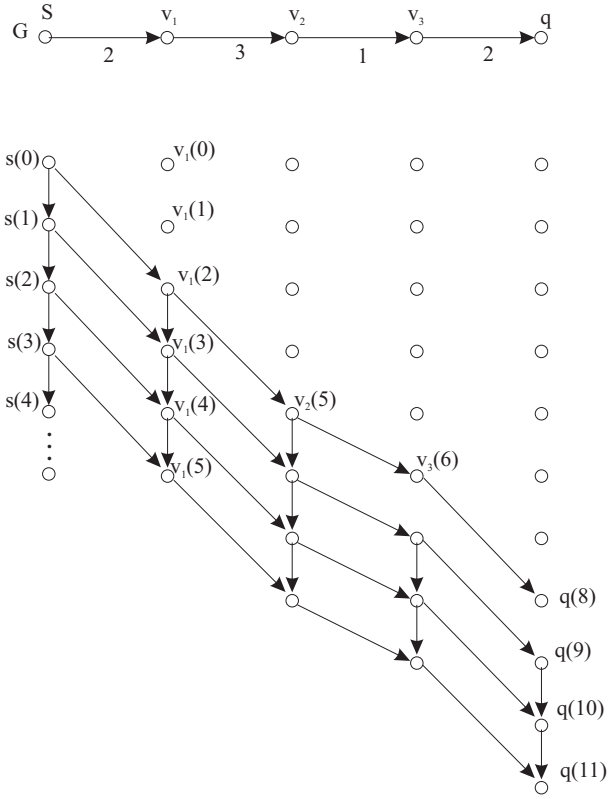


Рис. 1.

План перевозок для данной задачи имеет вид таблицы с T строками и n столбцами, т.е. план перевозок есть сокращенная запись ПВД. Заметим, что поток по ПВД является статическим, т.е. по каждому пути поток имеет мощность $c(\pi)$ и что перевозки можно организовать так, чтобы ни в одной вершине груз не задерживался. Например, до узкого места в пути π идут дуги с большой пропускной способностью, так что возникает мысль переправить максимальное количество груза в промежуточные вершины, а затем перевозить их по мере освобождения дуг. Можно показать, что справедливо следующее утверждение.

Утверждение 2. Задержки груза в промежуточных вершинах процесс перевозок не улучшают.

В общем случае, когда цепное разложение потока состоит из h путей, план перевозок получается сложением планов перевозок по отдельным цепям.

4. ДИНАМИЧЕСКИЙ ПОТОК

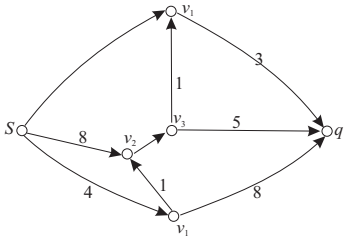
Из определения ПВД следует, что ее можно свернуть, заменив множество вершин $\{v_{t_0}, v_{t_0+1}, ...\}$ на вершину v и "параллельные дуги" вида $(v_{it}, v_{j,t+t_{i,j}})$ — дугой (v_i, v_j). Другими словами, мы возвращаемся к исходной сети, но уже с другой моделью перевозок, получаемой свертыванием статического потока (т.е. потока в обычном смысле) в *динамический поток*. Под последним мы понимаем набор потоков $f_1, f_2, \ldots, f_k (= f)$ с цепными разложениями $\pi_1, \pi_2, \ldots, \pi_k$. При этом существуют такие моменты времени $t_0, t_1, \ldots, t_k$, что

$$f_{din} = \begin{cases} 0, \text{ если } T < t_0; \\ f_1, \text{ если } t_0 \le T < t_1; \\ \vdots \\ f_k, \text{ если } T \ge t_k. \end{cases}$$

Моменты времени $t_0, t_1, \ldots, t_k$ определяются следующим образом. Пусть P_1 — кратчайший путь из s в q с множеством дуг E_{p_1}. По ним идет поток f_1. Пусть G_1 — остаточная сеть G_1, полученная из G удалением пути P_1 в том смысле, что дуги из пути P_1 заменяются дугами с пропускной способностью $c(i,j) - f(i,j)$. Заметим, что как минимум одна дуга будет удалена из-за своей насыщенности, т.е. для нее $C(i,j) = f_1$. Чтобы можно было решить нашу задачу, разрешаем проходить дуги с потоком $f(e) > 0$ в направлении, противоположном ее ориентации.

В остаточной сети G_1 отыскиваем кратчайший путь P_2 (точнее, кратчайшую цепь), так как мы будем проходить дуги с $f > 0$ в направлении, противоположном их ориентации. Эту задачу можно решить алгоритмом Дейкстры. В общем случае после отыскания цепи происходит автоматическая перестройка цепного разложения, в котором число отдельных путей становится на 1 больше и самый короткий путь в цепном разложении π_2 уже не совпадает с исходным. Однако следует заметить, что план перевозок для цепного разложения $\{\pi_1\}$ не изменяется и не влияет на окончательный результат. Длина кратчайшей цепи P_2 является моментом t_2 в определении динамического потока.

Пример. Пусть G имеет вид:



Динамический поток для сети на рис. 2 имеет вид:

Рис. 2.

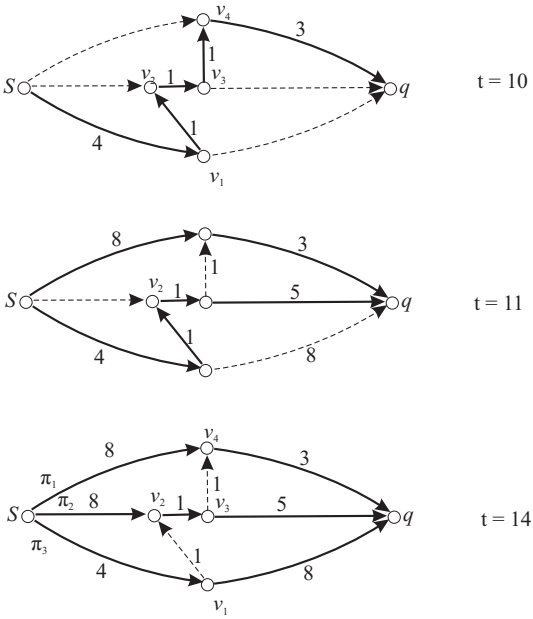


Рис. 3.

План перевозок для динамического потока f_1 / f_2 / f_3 показан на рис. 4.

t	S	v_1	v_2	v_3	v_4	q
1	$P-3$					
2	$P-6$					
3	$P-8$					
4	$P-10$					
5	$P-11$	1				
6		1				
7		1				
8		1			1	
9			1		1	
10			1	1	1	
11				1	1	1
12					1	3
13					1	5
14						8
15						11

Рис. 4.

Главное, что нужно отметить, это то, что динамический поток определяет возможность направить новые порции груза на дуги с ненулевым потоком. При изменении потока так, что часть порций груза осталась на дугах с измененным потоком, они заканчивают свое движение по таким дугам в прежнем режиме (к выходу сети).

Таким образом, изменение потока не вызывает уничтожения движущихся порций. И кроме того, можно не обращать внимание, по каким путям пересылались порции груза в самом начале перевозок. Это позволяет строить окончательный план перевозок на основе цепного разложения последнего использованного динамического потока.

5. МНОГОПОЛЮСНЫЕ СЕТИ

Рассмотрим теперь задачу организации перевозки однородного груза из входов $\{s_1, s_2, \ldots, s_n\}$ с запасами (или объемом производства), равными $\{P_1, P_2, \ldots, P_n\}$, в выходы $\{q_1, q_2, \ldots, q_m\}$ с потребностями

$\{Q_1, Q_2, \ldots, Q_m\}$. Без ограничения общности можно считать, что

$$\sum_{k=1}^{n} P_k = \sum_{l=1}^{m} Q_l.$$

5.1. Сети с одним входом

Рассмотрим сеть, показанную на рис. 5. Предположим, что пропускные способности всех дуг одинаковы и равны 1.

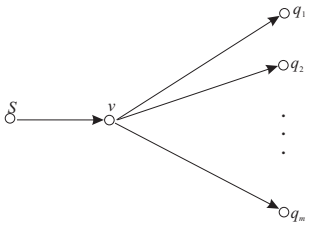


Рис. 5.

Длины дуг, исходящих из v, равны $\theta_1, \ldots, \theta_m$, причем предполагаем, что

$$\theta_1 \geq \theta_2 \geq \ldots \geq \theta_m.$$

Динамический поток, равный 1 на дуге (s, v), принимает ненулевое значение последовательно на дугах (v, q_1), (v, q_2) и т. д. В работе [6] при определении многополюсных потоков не было подчеркнуто, что указание только допустимых размеров потока без указания временных интервалов существования потоков на дугах вида (v, q_k) есть небрежность, запутывающая читателя.

С комбинаторным подходом к решению многополюсной транспортной задачи по времени можно ознакомиться в работе [7].

5.2. Многополюсные сети

Нетрудно убедиться, что решение транспортной задачи по времени для такой сети определяется на основе принципа “первым удовлетворяется спрос самого дальнего выхода” [2, 3].

В общем случае пары (s_i, q_j) находятся с помощью решения вспомогательной матричной транспортной задачи по времени, где величины t_{ij}

определяются из решения для каждой пары вход–выход двухполюсной транспортной задачи по времени. После этого решается общая задача как задача динамического программирования [1].

6. ЗАКЛЮЧЕНИЕ

Несмотря на близость рассматриваемой задачи к задаче о червеобразной посылке сообщений, они всё-таки отличаются, так как процесс перевозок включает в себя процесс рассыпания груза на порции, перемещающиеся по различным путям. Это требует обратной сборки сообщения. Больший прогресс можно достичь при планировании спонтанно возникающих червеобразных сообщений между полюсами в многополюсной сети. Здесь может оказаться полезной матричная транспортная задача по времени.

СПИСОК ЛИТЕРАТУРЫ

1. **Евстигнеев В.А.** Транспортная задача по времени для многополюсных сетей // Дискретный анализ. — 1966. — Вып. 6. — С. 9–34.
2. **Евстигнеев В.А.** Транспортная задача по времени.I. // Дискретный анализ. — 1968. — Вып. 13. — С. 3–20.
3. **Евстигнеев В.А.** Транспортная задача по времени. II. // Управляемые системы. — 1968. — Вып. 1. — С. 21–28.
4. **Евстигнеев В.А., Касьянов В.Н.** Толковый словарь по теории графов в информатике и программированию. — Новосибирск: Наука. Сиб. предприятие РАН, 1999.
5. **Форд Л.Р., Фалкерсон Д.Р.** Потоки в сетях. — М.: Мир, 1966.
6. **Евстигнеев В.А.** Многополюсные транспортные сети // Дискретный анализ. — 1965. — Вып. 4. — С. 28–36.
7. **Евстигнеев В.А., Фридман В.В.** Транспортная задача с древовидным графом маршрутов // Моделирование в экономических исследованиях. — Новосибирск: Изд-во Наука, СО АН СССР, 1978. — С. 84–95.

В. А. Евстигнеев, И. Л. Мирзуитова

РАЗВИТИЕ NUMA-АРХИТЕКТУРЫ: ТЕКУЩЕЕ СОСТОЯНИЕ*

1. ВВЕДЕНИЕ

Данная работа является продолжением исследований, начатых в [1].

Понятие NUMA-архитектуры возникло в конце 80-х годов с появлением интереса к компьютерам с распределенной памятью. Среди них выделялись так называемые DSM-машины, т. е. машины с распределенной общей памятью. В этих машинах общая память была физически разнесена по компьютеру. В силу этого сочетались легкость программирования для машин с общей памятью и серьезные трудности для поддержания режима общей памяти. При этом кусок общей памяти, максимально приближенный к узлу вычислительной системы, имеет время доступа в десятки раз меньше, чем к удаленным кускам. Заметим, что максимально приближенный к узлу кусок общей памяти не есть локальная память этого узла, хотя и имеет много общего с ней.

Как мы уже знаем из [1], такие машины относятся к машинам с NUMA-архитектурой, если они специально нацелены на максимальное сглаживание разницы во времени доступа к разным частям памяти.

Другая проблема, решение которой требует наличия распределенной памяти, есть проблема масштабирования. Разбитая на куски, разнесенная память позволяет организовать параллельные машины в виде сети пар процессор–память. К масштабируемым машинам этого типа относятся машины фирмы BBN (Batterfly, TC 2000, GP 2000), KSR-1 и KSR-2 фирмы Kendall Square Research, NCUBE 2, а также более поздние — IBM SP-2, Convex SPP 1200/XA, Intel Paragon, Thinking Machine CM5, IBM RP3, проект DASH (Стенфорд), проект ALEWIFE (МТИ), Horizon/Tera.

Но многие авторы утверждают, что NUMA-архитектура была разработана в начале 1990-х. На май 1998 г. она поддерживалась только несколькими поставщиками аппаратуры (например, Sequent Computers Inc., Data General Corp., Silicon Graphics Inc. и Hewlett-Packard) и систем баз данных. Это расходится с приведенным выше списком машин,

*Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

относимых к NUMA-архитектуре. Сторонники NUMA утверждают, что эта архитектура позволяет решить все проблемы, свойственные другим параллельным архитектурам. Однако так было всегда: каждый, кто предлагал новую параллельную архитектуру, заявлял, что она решает все старые проблемы. Вместе с тем, имеются основания ожидать, что архитектура NUMA останется среди реально используемых.

Следующим этапом развития NUMA-архитектуры является появление архитектуры ccNUMA — акроним от "cache-coherent nonuniform memory architecture". ccNUMA — это кэш-когерентный доступ к неоднородной памяти. Архитектура ccNUMA выделяется принципиально. Это архитектура симметричного мультипроцессирования (SMP), обладающая множеством достоинств: простая модель программирования, переносимость приложений и т.д. У истоков создания архитектуры ccNUMA стояла компания Sequent, реализовавшая собственную версию — NUMA-Q.

Следующим этапом развития архитектуры NUMA является появление архитектуры NUMAFlex. 25 июля 2000 г. компания SGI анонсировала новое семейство систем SGI 3000. Эти системы являются первыми системами, построенными на основе технологии NUMAFlex. Технология NUMAFlex, которая является третьим поколением технологии NUMA, дает возможность наращивания и изменения системы вплоть до использования различных процессоров одновременно в рамках единой системы. Не исключено, что технология NUMA станет открытой.

Основным блоком конструкции новых серверов SGI 3000 стал "кирпич" (brick); при этом кирпичи бывают разных типов, в зависимости от их содержимого. Однако основные элементы архитектуры S2MP сохранены, т.е. сохраняется то, как связываются между собой процессоры, оперативная память, концентраторы, маршрутизаторы и подсистема ввода-вывода. То, что ранее было реализовано в виде плат, "превратилось" в кирпичи, а "провода" на системной плате типа midplane заменены кабелями (таких плат в NUMAFlex больше нет).

2. ОСНОВНЫЕ ПАРАЛЛЕЛЬНЫЕ АРХИТЕКТУРЫ

Чтобы понять принципы работы технологии NUMA, нужно знать, как функционирует традиционная симметричная многопроцессорная обработка (SMP). SMP позволяет связать несколько процессоров в одну систему и объединить их вычислительную мощность для выполнения нескольких приложений или одного большого приложения. Эти процес-

соры взаимодействуют друг с другом с помощью так называемой шины межсоединения и используют пул общей памяти. При увеличении в сервере числа процессоров возрастает и трафик в данной шине. В конце концов пропускная способность системы существенно снижается.

NUMA, как и SMP, позволяет получить объединенную вычислительную мощность большого числа процессоров, каждый из которых обращается к общему пулу памяти, однако процессоры организуются в небольшие группы или "узлы", с помощью которых они связываются друг с другом. Архитектура SMP появилась в начале 1970-х гг. и быстро стала фактическим стандартом наиболее распространенных параллельных архитектур. Если компания, производящая компьютерное оборудование, поддерживает хотя бы какой-нибудь вид параллелизма, то с большой вероятностью используется SMP.

Кластеры, вероятно, стали использоваться даже раньше, чем SMP. Причиной возникновения кластерной архитектуры было то, что необходимую для пользователя работу было невозможно выполнить на одном компьютере или эта работа была настолько важна, чтобы приобрести дублирующее оборудование. Много позже этот подход удостоился общепринятого названия (термин "кластер" был введен в обиход компанией Digital Equipment Corp. в середине 1980-х гг.), и его стали поддерживать поставщики систем. Сегодня кластерная архитектура является козырной картой практически каждого поставщика компьютерных систем, ориентированных на применение ОС UNIX и/или Windows NT.

2.1. Архитектура SMP

SMP — это один компьютер с несколькими равноправными процессорами. Все остальное — в одном экземпляре: одна память, одна подсистема ввода/вывода, одна ОС. Слово "равноправный" (как и слово "симметричная" в названии архитектуры) означает, что каждый процессор может делать все, что любой другой. Каждый процессор имеет доступ ко всей памяти, может выполнять любую операцию ввода/вывода, прерывать другие процессоры и т.д. Но это представление справедливо только на уровне программного обеспечения. Умалчивается то, что на самом деле в SMP имеется несколько устройств памяти.

В SMP каждый процессор имеет по крайней мере одну собственную кэш-память (а возможно, и несколько). Наличие кэш-памяти (или просто кэша) необходимо для достижения хорошей производительности, поскольку основная память (DRAM —Direct Random Access Memry)

работает слишком медленно по сравнению со скоростью процессоров, и каждый год это соотношение ухудшается. Уже сейчас скорость основной памяти в 20–40 раз меньше, чем требуется, а в ближайшее время этот показатель будет порядка 100. Кэш работает со скоростью процессора, но эта аппаратура дорогая, и поэтому устройства кэш-памяти обладают относительно небольшой емкостью. В настоящее время размер кэш-памяти составляет 10–100 Кбт, в то время как основная память может иметь объем в 10–100 Мбт. Для процессора кэш исполняет роль "рабочего стола", на котором хранится используемая в текущее время информация: основная память подобна большому шкафу, находящемуся в комнате. При том, что в SMP имеется несколько устройств памяти, программное обеспечение ожидает увидеть только одну общую память. Грубо говоря, из этого следует, что если процессор A сохраняет значение X в ячейке Q, а позже процессор B загружает значение из ячейки Q, то процессор B должен получить X. Но если на самом деле значение X было помещено в кэш-процессор A, то как его сможет получить процессор B?

Имеются три причины, по которым когерентность кэшей является важной. Во-первых, это свойство играет ведущую роль в системах NUMA. Во-вторых, поддержка когерентности серьезно влияет на производительность. И, в третьих, именно из-за поддержки когерентности кэшей архитектура SMP не может обеспечить высокой доступности. Влияние на производительность очевидно. Программа работает гораздо быстрее, в 10–20 раз, если используются данные, уже содержащиеся в кэше. Программы, которые обращаются к другим устройствам памяти, выполняются очень медленно. Даже если для реорганизации кода с целью повышения вероятности использования данных из кэша требуется большее число команд, эти команды гораздо более быстрые.

2.2. Кластерная архитектура

Кластер — это связанный набор полноценных компьютеров, используемый в качестве единого ресурса. Под словосочетанием "полноценный компьютер" понимается завершенная компьютерная система, обладающая всем, что требуется для ее функционирования, включая процессоры, память, подсистему ввода/вывода, а также ОС, подсистемы, приложения и т.д. Обычно для этого годятся готовые компьютеры, которые могут обладать архитектурой SMP и даже NUMA.

Другое название кластерной архитектуры — мультикомпьютеры.

Словосочетание "единый ресурс" означает наличие ПО, дающего возможность пользователям, администраторам и даже приложениям считать, что имеется только одна сущность — кластер. Например, система пакетной обработки кластера позволяет послать задание на обработку кластеру, а не какому-нибудь отдельному компьютеру. Более сложным примером являются системы баз данных. У всех ведущих поставщиков систем баз данных имеются версии, работающие в параллельном режиме на нескольких машинах кластера. В результате приложения, использующие базу данных, не должны заботиться о том, где выполняется их работа. СУБД отвечает за синхронизацию параллельно выполняемых действий и поддержания целостности базы данных.

При соединении компьютеров в кластер почти всегда поддерживаются прямые межмашинные коммуникации. Решения могут быть простыми, основывающимися на аппаратуре Ethernet, или сложными с высокоскоростными сетями с пропускной способностью в сотни мегабайт в секунду. К последней категории относятся RS/6000 SP компании IBM, системы компании Digital на основе Memory Channel, ServerNet компании Compaq Computer Corp. Часто, хотя и не всегда, обеспечивается доступ каждого компьютера, входящего в состав кластера, к любому диску. Это не означает, что компьютеры обязательно совместно используют диски каким-либо разумным образом; при разработке СУБД или других подсистем возможен выбор, и в большинстве таких систем применяется подход "sharing–nothing", при котором в любой момент времени диском "владеет" только один компьютер. Исключениями являются Oracle8 и DB2 с IBM OS/390; в них диски активно используются в совместном режиме, образуя большую и медленную память, с помощью которой поддерживается когерентность кэшей базы данных.

2.3. Архитектуры NUMA

Проще всего охарактеризовать NUMA-систему, представив себе большую систему SMP, распиленную пополам, причем половинки связаны кабелем, подключенным к системным шинам, и каждая такая половинка включает собственную основную память и систему ввода/вывода. Это и есть NUMA: большая SMP, разбитая на набор более мелких и простых SMP. Основной проблемой NUMA является обеспечение когерентности кэшей. Аппаратура позволяет работать со всеми отдельными устройствами основной памяти составных частей системы (называемых обычно узлами) как с единой гигантской памятью. Этот подход порож-

дает ряд следствий. Во-первых, в системе имеется одно адресное пространство, распространяемое на все узлы. Реальный (не виртуальный) адрес 0 для каждого процессора в любом узле соответствует адресу 0 в частной памяти узла 0 и т. д., пока не будет использована вся память узла 0. Затем происходит переход к памяти узла 1, затем — узла 2 и т. д. Для реализации этого единого адресного пространства каждый узел NUMA включает специальную аппаратуру (Dir).

NUMA-системы вряд ли смогут заменить SMP при решении задач, требующих частой синхронизации. Ситуация улучшается, если удается разделить приложение на части, каждая из которых располагается в одном узле и взаимодействия между которыми возникают не слишком часто. Например, NUMA-систему с двумя узлами можно эффективно использовать для ведения бухгалтерии корпорации, имеющей два относительно независимых офиса. Эффективно использовать NUMA-системы могут СУБД, оптимизаторы которых в состоянии разбить сложный запрос на независимо выполняемые части (например, сканирующие разные части большой таблицы).

Что касается доступности, то NUMA наследует все неприятности, свойственные SMP, по той же причине обеспечения когерентности кэшей и наличия единой ОС. В отличие от кластеров, NUMA-систему любого размера можно считать “одной машиной”. Для достижения высокого уровня доступности нужно использовать кластеры.

3. ОПИСАНИЕ ПЛАТФОРМЫ NUMA-Q ОТ SEQUENT (АРХИТЕКТУРА CCNUMA)

ccNUMA — это кэш-когерентный доступ к неоднородной памяти. В системе ccNUMA физически распределенная память объединяется, как в любой другой SMP-архитектуре, в единый массив. Не происходит никакого копирования страниц или данных между ячейками памяти. Нет никакой программно-реализованной передачи сообщений. Существуют просто одна карта памяти, частями, физически связанными медным кабелем, и очень умные (в большей степени, чем объединительная плата) аппаратные средства. Аппаратно-реализованная кэш-когерентность означает, что не требуется какого-либо программного обеспечения для сохранения множества копий обновленных данных или для передачи их между множеством экземпляров ОС и приложений. Со всем этим справляется аппаратный уровень точно так же, как в любом SMP-узле, с одной копией ОС и несколькими процессами.

3.1. Общая структура

Элементарным блоком платформы NUMA-Q служит квод (NUMA-Q означает ccNUMA с кводами), в котором объединены четыре процессора, блок разделяемой памяти и шины PCI с семью слотами. Несколько кводов могут быть соединены связями с аппаратно-реализованной кэш-когерентностью для формирования более крупного одиночного SMP-узла таким же образом, как процессорные платы добавляются к объединительной плате обычного SMP-узла с большой шиной.

3.2. Задержка — как основная характеристика различия архитектур

Одним из ключевых различий описанных архитектур является диктуемая ими модель программирования, а различия в способах программирования напрямую обусловлены задержками доступа. Достижение организации памяти NUMA-Q заключается в том, что доступ к часто используемым данным происходит за микросекунды, тогда как считывание их с диска требует миллисекунд, а в MPP-системах с разделяемыми дисками доступ к удаленному диску может занимать десятки миллисекунд. В кластерах с отражением памяти между узлами добавляются когерентные соединения с программной поддержкой, в результате чего время доступа к удаленной памяти снижается до сотни микросекунд. Это, конечно, в сотни раз быстрее, чем обращаться к дискам для тех же данных, но сотни микросекунд — это еще в сотни раз медленнее, чем скорость локальной памяти. Поэтому программист должен позаботиться о том, чтобы минимизировать такого рода пересылки, планируя для этого, где только возможно, распределение данных вручную. Следует отметить, что даже в RMC удаленный доступ опирается как на аппаратную, так и на программную поддержку, тогда как доступ к локальной памяти реализуется исключительно аппаратными средствами — очень быстро и с гарантией когерентности. Вот почему программирование в SMP так выгодно для прикладных программистов, которые не должны заботиться о распределении данных в памяти, так как все ее части доступны для любого ЦПУ и доступ к ним одинаково быстр.

3.3. Порядок работы NUMA-Q

В качестве основного строительного блока для SMP-узлов платформы NUMA-Q компания Sequent использует кводы с четырьмя процес-

сорами на блок распределенной памяти. В узле с тремя, например, квадами одна треть физической памяти будет близка (в смысле задержки доступа к памяти) к четырем процессорам квада, а две трети будут "не совсем близкими". Это может привести к выводу о том, что две трети обращений к памяти будут медленными и только одна треть — быстрой. К счастью, без модификации приложений, реализованных для традиционных SMP-архитектур, этого не происходит. В действительности основная часть процессорных доступов к памяти будет очень быстрой, и только маленький процент окажется не столь высокоскоростным. Это происходит из-за большой локальной памяти квада и большого удаленного кэша на плате IQ-Link.

3.3.1. Конфигурация памяти NUMA-Q

В традиционном смысле память в каждом кводе не является локальной. Скорее, это одна треть адресного пространства физической памяти, которая имеет собственный адресный диапазон. Адресная карта распределяется по памяти равномерно, при этом каждый квод содержит смежную часть адресного пространства. Как и в любой SMP-системе, работает только одна копия ОС, она размещает в памяти и запускает одновременно на одном или нескольких процессорах любые процессы без какого-либо различия между ними.

Будем называть сегмент памяти, расположенный в кводе, локальной памятью квода, а память из других квадов — удаленной памятью квода. Ясно, что доступ к локальной памяти квода происходит быстрее, чем доступ к удаленной для него памяти. Задержки доступа к единому пространству адресов памяти не одинаковы, вот почему NUMA-Q является истинной NUMA-архитектурой.

В современной реализации архитектуры NUMA-Q компания Sequent использует процессоры Pentium Pro с двумя кэшами L1 и L2 внутри чипа. Как известно, в компьютерных системах кэши устанавливаются в предположении о "пространственной локальности" обращений к памяти. В соответствии с этим предположением основная часть всех кэш-промахов (cache miss) в L2 будет попадать в диапазон локальной памяти квода и, таким образом, будет быстро обслуживаться. Если же адрес находится за пределами диапазона локальной памяти квода, поиск будет распространен на 32-мегабайтный кэш IQ-Link, который называется удаленным кэшем. Доступ к этому кэшу осуществляется с такой же скоростью, как и к локальной памяти квода.

3.3.2. Кэш-когерентность в NUMA-Q

Передача данных по единственной шине в SMP-архитектурах с одной объединительной платой может происходить по разным причинам, и их следует различать. Во-первых, это передача между портом ввода-вывода и памятью, и, во-вторых, ситуации, в которых процессор обращается к памяти при отсутствии данных в L2 кэше. Кроме того, имеются еще и передачи кэш-кэш между разными процессорами, которые называются промахами когерентности.

3.3.3. Нужно ли оптимизировать программы для NUMA-Q?

Эффективность работы приложения на платформе NUMA-Q зависит, главным образом, от того, насколько справедливы следующие предположения.

1. Частота обращений к удаленной памяти существенно ниже, чем частота локальных обращений.
2. Задержка удаленного доступа очень мала.
3. Пропускная способность IQ-Link намного больше, чем та, которая требуется в приложениях OLTP и DSS.

Что должны делать разработчики ПО для систем SMP при переносе его на NUMA-Q? Если некоторые из приведенных пунктов неверны, можно попытаться изменить ПО, чтобы настроить его на NUMA-Q. Однако, если предположения выполняются, тогда тот факт, что малый процент доступов занимает больше времени, чем остальные, может не учитываться программистами так же, как не принимается во внимание работа кэшей.

3.4. Заключение

Определяющим фактором для производительности систем являются задержки доступа к данным. Для уменьшения их влияния в системы вводится дополнительная память. SMP-платформы обеспечили легко программируемую модель, так как время доступа ко всем частям памяти стало одинаково недолгим. Попытки распределять данные между узлами, передавая их между памятью, успешны только в том случае, если альтернативой являются обращения за данными к диску. Однако для оптимизации производительности все еще требуется существенный объем перепрограммирования, так как задержки между узлами значительно больше, чем задержки внутри узла.

Наиболее эффективный способ достичь высокой производительности и сохранить при этом простую модель программирования — построить как можно больший одиночный узел, прежде чем переходить к архитектуре из нескольких узлов. Однако из-за ограничений на размер объединительных плат и системных шин, с использованием кэша слежения этого сделать нельзя, и максимально достигнутое число процессоров не превышает сегодня 32. Для того чтобы выйти за этот предел, можно использовать кэш-протоколы на основе каталогов и архитектуры ccNUMA. Такой подход имеет дополнительные достоинства, так как позволяет подсоединять до 252-х процессоров, получая огромную память и пропускную способность шины ввода-вывода и при этом имея средние задержки доступа к памяти меньше, чем у любой современной системы. Основным же достижением является отсутствие необходимости менять ПО SMP-приложений для того, чтобы воспользоваться всеми этими возможностями.

4. NUMAFLEX-АРХИТЕКТУРА

4.1. Введение

Напомним, что система ccNUMA состоит из набора узлов, каждый из которых имеет собственные процессоры, локальную оперативную память и обычно собственные средства ввода-вывода. Это справедливо и для произвольной MPP-системы с распределенной между узлами оперативной памятью, например, для кластерной архитектуры IBM SP2. Следующим шагом являются NUMA-системы, в которых память попрежнему физически распределена между узлами, но адресуема всеми микропроцессорами и логически является общей. Примером такой системы является Cray T3E. Наконец, для автоматического обеспечения согласованности работы всех процессоров с памятью требуется поддержание когерентности их кэшей, что и приводит разработчиков к архитектуре ccNUMA.

Похоже, что именно в направлении ccNUMA архитектура многопроцессорных систем развивается наиболее активно. Следующие ведущие производители предлагают компьютеры с архитектурой ccNUMA: это и HP (еще со времен Convex SPP), и Compaq с новыми компьютерами AlphaServer GS 320, и IBM Sequent NUMA-Q, и Data General AViiON 2x00, и Siemens RM600E, и, наконец, SGI серверы Origin, которые стали несколько лет тому назад основным полигоном практического освоения ccNUMA.

SGI, которая при разработке Origin 2000 опиралась на результаты совместного со Стэнфордским университетом проекта DASH, имеет, как представляется, наибольший опыт в этой области: летом 1999 г. компания представила уже второе поколение ccNUMA — систему Origin 3x00. Архитектура их предшественников Origin 2000 называлась S2MP (Scalable Shared memory MultiProcessing); архитектура же Origin 3x00 носит название NUMAFlex.

Здесь flex, очевидно, есть сокращение от английского flexibility ("гибкость"). NUMAFlex действительно отличается особой гибкостью в построении различных конфигураций системы и ее изменении "на лету" в процессе реального функционирования. В NUMAFlex реализована возможность разбиения всей ccNUMA-системы на разделы (partition), которые являются более "мелкими" ccNUMA или SMP-компьютерами. (Схема разбиения, или парционирования, будет рассмотрена ниже.) Парционирование позволяет преображать ccNUMA-систему в кластерную структуру. Узлами этого кластера могут быть опять-таки ccNUMA-серверы.

4.2. Системы IBM высокой доступности

В конце 90-х гг. фирма IBM приступила к выпуску современных центров обработки данных. Это третье поколение технологии "non-uniform memory access" (NUMA), впервые представленной в 1996 г. компанией Sequent Computer Systems, Inc. К таким центрам относятся две системы: NUMA-Q 2000 и NUMACenter.

4.2.1. Сервер NUMA-Q 2000

NUMA-Q 2000 — это современный центр обработки данных с лидирующей в индустрии производительностью. Серверы предприятия NUMA-Q 2000 базируются на архитектурных решениях, которые обеспечивают высочайшую производительность, доступность и управляемость, что и требуется для обработки огромных информационных массивов и круглосуточной поддержки большого количества запросов. NUMA-Q обладают широкими возможностями.

4.2.2. Сервер NUMACenter

Весной 1999 г. Sequent представила упрощенный вариант серверов — NUMACenter, масштабируемых до 64-х процессоров. Тогда же был про-

демонстрирован режим работы этой системы под одновременным управлением UNIX и Windows NT, интегрированных с помощью Unicenter TNG компании CA. Сервер NUMACenter создан на базе новых процессоров Xeon.

IBM NUMACenter — идеальное решение для эффективной работы с коммерческими приложениями в информационной среде предприятия.

4.2.3. Серверы SGI

Серверы SGI — рыночные лидеры в технических вычислительных приложениях — используются в ключевых отраслях промышленности, в правительстве, индустрии развлечений, связи, энергетики, науки и образовании. Высокомасштабируемые сервера компании также имеют постоянно возрастающее присутствие на коммерческом рынке, с акцентом на стратегический деловой анализ, приложения INTERNET и обслуживание средств мультимедиа.

Линия серверов SGI, начинаясь с серверов начального уровня SGI 1100, 1200, 1450, включает сервера среднего уровня SGI Origin 200, Origin 200 GIGAchannel, SGI 2100, 2200, 3200, 3200C и заканчивается серверами высшего уровня SGI Origin 2400, Origin 2800, Origin 3400 и Origin 3800.

Сервера начального уровня имеют архитектуру SMP, от 1 до 2 CPU (1100 и 1200) или от 1 до 4 (1450); в качестве процессоров используются Intel Pentium III или Pentium III Xeon.

Все сервера среднего уровня имеют архитектуру NUMA, от 1 до 4 CPU (Origin 200), от 2 до 8 CPU (Origin 2100, 2200, 3200); в качестве процессоров используются MIPS R12000.

Сервера высшего уровня имеют архитектуру NUMA; Origin 2400 имеет от 2 до 512 CPU, Origin 3400 — от 4 до 32 CPU и Origin 3800 имеет от 16 до 512 CPU; в качестве процессоров используются MIPS R12000.

4.2.4. SGI 3000

25 июля 2000 г. компания SGI анонсировала новое семейство систем SGI 3000. Эти системы являются первыми системами, построенными на основе технологии NUMAFlex.

Данный компьютер построен на базе 64-битового процесора Itanium корпорации Intel. Кроме того, он будет работать под управлением ОС

Linux. Системы SGI 3000 имеют модульную архитектуру, которая позволяет строить системы, полностью отвечающие требованиям любых специализированных задач за счет разделения компонент компьютера на независимые функциональные блоки. По планам компании новый компьютер на базе Intel-Linux будет выпущен сразу же после начала выпуска компанией Intel процессоров Itanium. К сожалению, сроки появления этого процессора постоянно отодвигаются, и когда именно это произойдет, пока до конца не ясно.

4.2.5. Архитектура Origin 3x00

Архитектура NUMAFlex очень близка к той, что была использована в Origin 2000, т.е. S2MP. Большая часть усовершенствований связана с конструктивными изменениями: в Origin 3x00 резко увеличена модульность и надежность системы.

Основным блоком конструкции новых серверов стал "кирпич" (brick); при этом кирпичи бывают разных типов, в зависимости от их содержимого. Однако основные элементы архитектуры S2MP сохранены, т.е. сохраняется то, как связываются между собой процессоры, оперативная память, концентраторы, маршрутизаторы и подсистема ввода-вывода. То, что ранее было реализовано в виде плат, "превратилось" в кирпичи, "провода" на системной плате типа midplane заменены кабелями (таких плат в NUMAFlex больше нет).

Наряду с C-кирпичами используются другие кирпичи (всего 6 типов).

Очень важной особенностью архитектуры NUMAFlex, определяемой применением кирпичей, является исключительно высокая гибкость в построении различных конфигураций и сохранение инвестиций пользователя при модернизации. Заказчик приобретает только те кирпичи, которые ему действительно нужны, и "складывает" из них компьютер нужной конфигурации (конечно, сначала компьютер собирают все-таки на заводе). В случае же приобретения многопроцессорных систем, использующих конструктив общей системной шины на платах типа backplane или midplane, платить приходится за каждую такую плату со всеми расположенными на ней компонентами. Конкретной иллюстрацией такой гибкости может служить проведенное выше сопоставление Origin 3x00 и 2000 в части поддержания слотов XIO. С другой стороны, наращивание конфигурации минимальными порциями по 4 процессора, что, естественно, достаточно дорого, выбивается из общей картины.

5. NEC CENJU-4

В этом разделе мы дадим, следуя работе [2], краткий обзор ЭВМ NEC Cenju-4 — платформы для разнообразных построений.

5.1. Краткий обзор

Cenju-4 есть параллельный компьютер, построенный и выпускаемый компанией NEC.

Cenju-4 представляет собой мультипроцессор с архитектурой NUMA. Многоступенчатая сеть этой машины связывает до 1024 узлов, используя 4×4-шаговый коммутатор. Сеть имеет следующие особенности: симметричная (in-order) посылка сообщений между двумя любыми узлами, функции группировки и рассыпания, механизм, свободный от дедлоков. Каждый узел Cenju-4 состоит из процессора R10000, 1 Мб вторичного кэша, главной памяти объемом 512 Мб и чипа контроллера. Чип контроллера допускает посылку сообщений на уровне пользователя и DSM-доступ. Не допускается использовать и посылку сообщений, и DSM-доступ на одной и той же странице. Этот атрибут управляется ОС, которая основывается на микроядрах MACH per page bases.

5.2. Распределенная общая память

DSM ЭВМ Cenju-4 реализуется путем использования когерентных кэшей и основанных на директивах протоколах когерентности. DSM имеет следующие четыре характеристики.

- Директория динамически переключает ее представление со структуры указателей на структуру побитового шаблона согласно числу узлов. Эта схема требует постоянной области памяти независимо от числа процессоров, достигая эффективной записи узлов. Это дает масштабирование в аппаратуре и производительности.
- DSM в Cenju-4 использует функции группировки и рассыпания сети для доставки запросов и накапливания ответов. Это уменьшает накладные расходы на сообщения о недопустимости кэша. Cenju-4 также принимает директорию, которая может специфицировать все узлы, кэшируя (cashing) блок с одним доступом к памяти.
- Протокол когерентности кэша, который мешает “зависанию”. Cenju-4 принимает блокирующий протокол для когерентности

кэшей: запросы, которые не могут быть обработаны немедленно, помещаются в очередь в главной памяти для более поздней обработки. Размер буфера этой очереди — 32 Кб для 1024 узлов.

- Свободный от дедлоков механизм с одной сетью. Предлагается механизм, который образует очередь некоторых типов сообщений для когерентности кэша в главной памяти. Размер буфера, требуемого для создания очереди сообщений, равен 128 Кб для 1024 узлов. Этот буфер размещается в области, отличной от предыдущего буфера. Этот буфер и предыдущий для зависания размещаются и используются в различных функциях. Протокол когерентности кэша и свободный от дедлоков механизм гарантируют доступы к общей памяти с окончанием в конечное время.

Пользователи должны добавить вызовы библиотеки, чтобы использовать DSM-функции, так как компилятор, который может генерировать код для утилизации DSM-функции, не доступен. Общее адресное пространство размещается путем использования вызова библиотеки, и общие переменные размещаются или переразмещаются в таком пространстве. В будущем размещенные общие переменные могут быть доступными, так же как и приватные данные.

Существуют некоторые ограничения на использование DSM в Cenju-4: во-первых, наибольший объем общего адресного пространства ограничен объемом физической памяти. Далее, объем общего адресного пространства ограничено 2 Гб. Это объясняется тем, что используется архитектура с MIPS процессором, которая ограничивает адресное пространство пользователя 2 Гб.

6. ЗАКЛЮЧЕНИЕ

При написании данной статьи мы использовали доступные материалы из сети Internet. При этом принимались во внимание как электронные версии статей, так и информационные и рекламные материалы. Были использованы статьи И. Бородина “Архитектура современных суперкомпьютеров” (2001 г.), Е. Коваленко “Система Sequent NUMA-Q” (1997), М. Мосейкина “Параллельные системы и кластеры: проблема выбора” (1998), М. Сонгини, Д. Коннора “Возможные перспективы архитектуры NUMA” (1998), М. Кузьминского «“Кирпичные” компьютеры». Серверы нового поколения архитектуры NUMA компании SGI (2000) и др.

СПИСОК ЛИТЕРАТУРЫ

1. **Евстигнеев В. А.** NUMA-архитектура: некоторые особенности компиляции и генерации кода // Поддержка супервычислений и Интернет-ориентированные технологии — Новосибирск: ИСИ СО РАН. 2001. — С. 44–53.
2. **Kusano K., Sato M., Hosomi T., Seo Y.** The Omni OpenMP Compiler on the distributed shared memory of Cenju-4 // Lect. Notes Comput. Sci. — 2001. — Vol. 2104. — P. 20–30.

М. А. Иванов

ОБЗОР MPEG-ПОДОБНЫХ МЕТОДОВ КОДИРОВАНИЯ ВИДЕОДАННЫХ*

ВВЕДЕНИЕ

С возникновения первых цифровых видеосистем началась активная разработка методов и алгоритмов сжатия видеоданных. Для качественной оцифровки телевизионного сигнала шириной 6 МГц необходимо работать с частотой дискретизации не менее 12 МГц (теорема Котельникова), т. е. брать как минимум 12 млн. отсчетов в секунду. При использовании восьмибитного кодирования получим скорость передачи сигнала яркости (для черно-белой картинки) 100—120 Мбит/с. Добавим сюда цветовую информацию и служебные сигналы, получим скорость 270 Мбит/с. Обработать в реальном времени, передать и сохранить такой поток зачастую не под силу даже современным компьютерным системам.

В 1988 г. немногочисленная группа молодых экспертов (MPEG — Motion Picture Expert Group) взялась за разработку формата хранения видеоинформации на CD-ROM и ее воспроизведения со скоростью около 1,5 Мбит/с. В январе 1992 г. опубликованы спецификации MPEG-1 — первого представителя семейства MPEG. В качестве стандарта они были приняты почти через два года, к декабрю 1993 г.

Логическим продолжением MPEG-1 стал формат MPEG-2, разработанный не столько для хранения, сколько для передачи данных. Ключевой особенностью MPEG-2 является возможность разделить результирующий, подготовленный к передаче, видеосигнал на несколько независимых потоков, содержащих сигналы различного качества.

С приходом идеи телевидения высокой четкости (High Definition TV — HDTV) началась работа над форматом MPEG-3. Впоследствии оказалось, что MPEG-2 (с некоторыми уточнениями) вполне пригоден для этих целей, и работы над MPEG-3 были прекращены.

Зато начались работы над MPEG-4 — принципиально новым алгоритмом компрессии и передачи цифровых аудио- и видеосигналов, предназна-

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

ченным для «кодирования аудио- и видеоданных с очень малыми скоростями передачи». И хотя разработка его спецификаций началась еще до массового признания Всемирной паутины, MPEG-4-подобные алгоритмы кодирования занимают лидирующее положение среди методов Internet-вещания.

В данной статье мы рассмотрим основные алгоритмы сжатия видеоданных в MPEG-подобных системах кодирования. Следует заметить, что эта статья ни в коей мере не является описанием какого-либо MPEG-подобного формата (подробные описания форматов можно взять на сайте www.mpeg.org). Здесь собраны лишь основные методики их построения. Способы кодирования аудиоданных также не будут рассмотрены.

1. ТИПЫ КАДРОВ

Все MPEG-подобные методы видеокодирования выделяют из видеопоследовательности (последовательности кадров) кадры нескольких типов. К каждому типу кадра применяется свой алгоритм кодирования. Основные типы кадров таковы:

- I-Frame — ключевые кадры (intra-frames);
- P-Frame — кадры, при кодировании/декодировании которых используется предыдущий кадр (predicted frames);
- B-Frame — кадры, при кодировании/декодировании которых используется как предыдущий, так и последующий кадр видеопоследовательности (bi-directional frames).

Рассмотрим каждый тип кадра подробнее.

I-Frame — так называемые ключевые или опорные кадры. Кодирование/декодирование кадров этого типа происходит без участия других кадров видеопоследовательности. Опорные кадры кодируются с максимально возможным качеством, т.к. по ним будут восстанавливаться последующие кадры, следовательно, они имеют больший размер по сравнению с другими кадрами, поэтому нежелательно иметь большое число опорных кадров в видеопоследовательности. Ключевые кадры, как правило, расставляются через определенные промежутки времени, поскольку декодирование произвольного кадра требует декодирования опорного кадра, т.е. если требуется декодировать какой-то кадр, то нужно декодировать ближайший предшествующий ему опорный кадр, а затем все кадры между опорным и требуемым кадром. Также ключевые кадры вставляются в моменты сильного изменения изображения по причинам, которые будут изложены ниже.

P-Frame — кадры, построенные на основе предыдущего кадра с использованием компенсации движения объектов сцены. Это наиболее часто встречающиеся кадры в кодированной видеопоследовательности. Их размер существенно меньше размера ключевых кадров. При их построении проводится сравнение декодированного предыдущего кадра и исходного текущего кадра. Производится попытка обнаружить похожие области в обоих кадрах, чтобы выяснить, как изменились местоположения объектов сцены. Такие изменения записываются в форме векторов движения, по которым будет восстановлен текущий кадр.

B-Frame — кадры, построенные на основе предыдущего и последующего кадров видеопоследовательности. По аналогии с P-Frame находятся вектора движения для перемещения объектов в моменты времени между предыдущим и текущим кадрами, а также между текущим и последующим кадрами. Далее возможны два способа построения: либо хранятся оба вектора движения, и по ним интерполяцией восстанавливается текущий кадр, либо вектора вообще не хранятся и восстановление происходит по взвешенной сумме векторов движения из предыдущего и последующего кадров. В первом случае размер кадра получается больше, чем размер P-Frame, но качество изображения заметно лучше, во втором случае размер кадра получается на порядок меньше, но качество при этом существенно хуже.

Поясним порядок расстановки, порядок кодирования/декодирования и порядок следования кадров в закодированной видеопоследовательности на рис. 1.

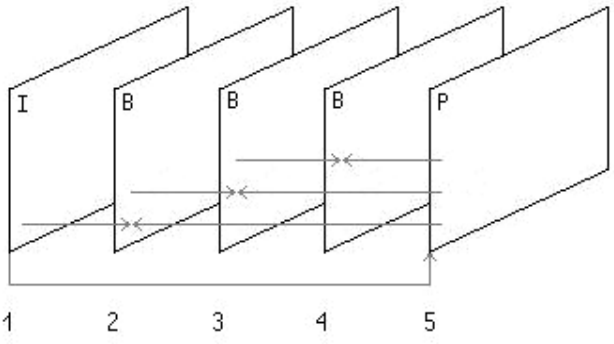


Рис. 1. Расстановка I-, P- и B- кадров, их зависимость друг от друга

Порядок кодирования/декодирования кадров в видеопоследовательности, представленной на рис. 1, следующий: сначала кодируется кадр 1 — ключевой кадр. Затем кодируется кадр 5 — P-Frame, далее, принимая кадр 1 за предыдущий, а кадр 5 — за последующий, кодируется кадр 2 — B-Frame, затем предыдущим кадром считаем кадр 2, а последующим — по-прежнему кадр 5, повторяем процедуру кодирования B-Frame.

В закодированной видеопоследовательности кадры должны следовать в порядке, допускающем их декодирование. Следовательно, первым в закодированной последовательности будет записан ключевой кадр 1, затем P-кадр 5, затем по порядку B-кадры 2, 3, 4.

При декодировании первым будет декодирован и отображен ключевой кадр 1, затем будет декодирован, но не отображен P-кадр 5, далее будут по порядку декодированы и отображены кадры 2, 3, 4 и после этого будет отображен кадр 5. Таким образом, возникнет небольшая задержка, равная времени декодирования кадра (~1/30 секунды), она может быть скомпенсирована задержкой в самом начале видеопоследовательности.

Методика выбора типа кадра может быть различной. Например, можно сравнивать так называемые среднеквадратичные разности кадров[1]:

$$MSE(F_1, F_2) = \frac{1}{w * h} \sum_{y=1}^{h} \sum_{x=1}^{w} \sqrt{(F_1(x,y) - F_2(x,y))^2} ,$$

где $F_k(x,y)$ — значение точки с координатами (x,y) в k-м кадре, h и w — соответственно высота и ширина кадра.

Можно также сравнивать максимальную по модулю разность кадров:

$$MAD(F_1, F_2) = \max_{y=1\ldots h, x=1\ldots w} \left| F_1(x,y) - F_2(x,y) \right| .$$

При использовании первого и второго способов экспериментальным путем определяются пороговые значения для каждого типа кадра и, в соответствии с полученным значением меры, выбирается типа кадра. Третий и наиболее эффективный способ определения типа кадра связан с поиском векторов движения и будет изложен ниже.

[1] Здесь и далее мы будем рассматривать только черно-белые видеопоследовательности. Цветные изображения могут быть рассмотрены как набор из трех черно-белых компонент, например в формате RGB или YCrCb. В последнем случае следует учесть, что яркостная компонента Y имеет большую значимость, чем цветовые компоненты Cr и Cb.

2. КОДИРОВАНИЕ КЛЮЧЕВЫХ КАДРОВ

Основными операциями при кодировании/декодировании ключевых кадров в MPEG-подобных видеокодерах являются операции прямого и обратного дискретного косинусного преобразования, а также операция квантования.

Дискретное косинусное преобразование (ДКП) является разновидностью преобразования Фурье. Оно позволяет перейти от пространственного представления изображения к его спектральному представлению и обратно. Результат ДКП для функции *F(x,y)*, заданной матрицей значений на дискретном множестве $\{1 \le y \le N, 1 \le x \le N\}$, есть матрица частотных коэффициентов *D* размера *NxN*.

Схема вычисления коэффициентов матрицы *D* определяется следующим выражением:

$$D(i,j) = \frac{1}{\sqrt{2N}} W(i) W(j) \sum_{y=1}^{N} \sum_{x=1}^{N} F(x,y) \cos\left(\frac{(2x+1)\pi i}{2N}\right) \cos\left(\frac{(2y+1)\pi j}{2N}\right),$$

где

$$W(i) = \begin{cases} \frac{1}{\sqrt{2}}, & i = 0; \\ 1, & i > 0. \end{cases}$$

Выражение для обратного преобразования матрицы гармоник, применяемое при распаковке изображения, записывается в виде:

$$F(x,y) = \frac{1}{\sqrt{2N}} \sum_{i=1}^{N} \sum_{j=1}^{N} W(i) W(j) D(i,j) \cos\left(\frac{(2x+1)\pi i}{2N}\right) \cos\left(\frac{(2y+1)\pi j}{2N}\right).$$

Для применения ДКП кадр разбивается на квадратные блоки (как правило, размера 8x8 точек), и затем преобразование применяется к каждому блоку в отдельности.

Время вычисления ДКП определяется по таким формулам $O(N^2)$, причем все операции должны быть проведены в числах с плавающей запятой, что существенно замедляет вычисления и, следовательно, скорость кодирования/декодирования. На практике применяется метод вычисления коэффициентов ДКП через стандартные операции умножения матриц. Схема

вычисления частотных коэффициентов выглядит таким образом: $D = C \cdot F \cdot C^T$, где

$$C(i,j) = \begin{cases} \dfrac{1}{\sqrt{N}}, & i = 0 \\ \sqrt{\dfrac{2}{N}} \cos\left(\dfrac{(2j+1)\pi i}{2N}\right), & i > 0. \end{cases}$$

Для перехода от чисел с плавающей запятой к целым числам используется умножение всех коэффициентов на подходящую степень двойки. После вычисления коэффициентов ДКП происходит обратное деление (умножение и деление на степень двойки может быть очень быстро реализовано на уровне регистров процессора с помощью операций побитового сдвига влево и вправо).

После вычисления коэффициентов ДКП следует этап их квантования. Этот этап нужен для подавления высокочастотных составляющих сигнала, которые менее заметны для человеческого глаза, чем низкочастотные, а также для уменьшения избыточности информации, что позволяет добиться более высоких коэффициентов сжатия. Квантование выглядит как поэлементное деление матрицы коэффициентов ДКП D на матрицу квантования Q. Элементы матрицы Q увеличиваются в сторону увеличения индексов. Например:

$$Q = \begin{pmatrix} 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 \\ 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 \\ 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 \\ 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 \\ 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 \\ 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 \\ 15 & 17 & 19 & 21 & 23 & 25 & 27 & 29 \\ 17 & 19 & 21 & 23 & 25 & 27 & 29 & 31 \end{pmatrix}$$

В реальных кодерах используется не одна матрица квантования, а некоторое их множество (обычно 100 матриц). Это позволяет регулировать качество сжатого изображения и степень компрессии — чем больше значения коэффициентов квантования, тем хуже качество и тем выше степень компрессии.

После такой операции амплитуды высокочастотных составляющих сигнала, находящиеся ближе к правому нижнему углу матрицы коэффициентов, обратятся в нули, что существенно повысит коэффициент сжатия. Для того чтобы учесть большое количество нулей в правом нижнем углу матрицы отквантованных коэффициентов, используется специальный обход «зигзаг» в сочетании с *run-length* кодированием.

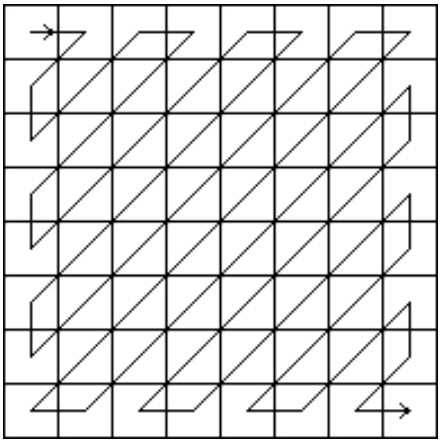


Рис. 2. Порядок обхода "зигзаг"

Суть *run-length* кодирования в том, что при обходе зигзагом будут встречаться длинные последовательности, состоящие из нулей. Нет смысла записывать каждый ноль. Выгодней записывать битовый флаг — ноль/не ноль, за которым следует либо количество нулей в последовательности, либо ненулевое значение коэффициента.

3. КОМПЕНСАЦИЯ ДВИЖЕНИЯ

Компенсация движения (motion compensation) в MPEG-подобных кодерах основана на так называемом предсказании движения (motion estimation) и кодировании остаточного изображения (residual).

Для предсказания движения кодируемый кадр разбивается на некоторые регулярные области (обычно это квадратные блоки размером 8x8, 16x16, 32x32, 64x64 точки). Далее для каждой такой области пытаются найти ее прообраз в предыдущем и/или последующем кадрах и записывают смещение этой области в виде вектора движения. Затем для получения остаточно-

го изображения найденный блок вычитается из блока в текущем кадре. Метод кодирования полученной разности будет рассмотрен далее.

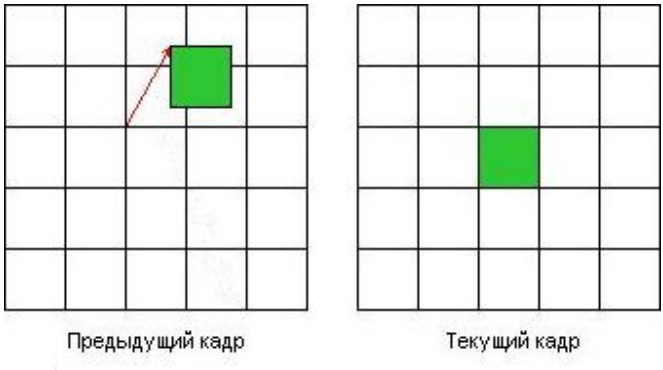


Рис. 3. Вектор движения для блока текущего кадра

На рис. 3 приведен пример нахождения вектора движения для одного блока текущего кадра в предыдущем кадре.

Исходя из соображений оптимизации поиска вектора по скорости, не имеет смысла искать вектор для текущего блока во всем предыдущем кадре. Обычно ограничиваются некоторой областью поиска, так называемый макроблок, представляющий собой квадратный блок, размеры которого в несколько раз превышают размер самого блока. Размеры макроблока могут меняться от 32x32 до 256x256. Нетрудно заметить, что чем больше размеры макроблока, тем более быстрые движения можно предсказать подобным методом, соответственно, тем медленней будет происходить поиск вектора движения.

Существуют несколько вариаций алгоритма поиска вектора движения. Они различаются скоростью работы и качеством предсказания. Рассмотрим наиболее часто употребляемые алгоритмы.

Полный перебор. Из названия метода ясно, что при его использовании перебираются все возможные варианты векторов движения. Порядок перебора следующий: фиксируем одну из координат вектора и перебираем все возможные значения второй координаты, затем меняем первую координату и т.д. Очевидно, таких векторов $(S_M - s)^2$ штук, где S_M — размер макроблока, s — размер блока. Далее из всех возможных векторов выбирается тот, который минимизирует разность блока и его прообраза (так называе-

мая ошибка поиска вектора движения) по некоторой мере. Обычно используется одна из мер, приведенных выше: MSE, MAD. Зачастую пользуются модификацией этого алгоритма, в которой перебор векторов заканчивается по достижении некоторого малого значения ошибки поиска вектора. Это существенно ускоряет время поиска.

Перебор по спирали. В этом алгоритме также перебираются все возможные вектора, но порядок перебора векторов таков, что конец вектора движения описывает расходящуюся или сходящуюся спираль вокруг блока. Перебор заканчивается по достижению порогового значения ошибки поиска. Преимущество этого метода в том, что если имеется какая-то информация о скоростях движения объектов в рассматриваемой области, то, выбрав правильное направление спирали (расходящаяся для медленных движений или сходящаяся для быстрых), можно достаточно быстро получить вектор с небольшой ошибкой поиска.

Логарифмический поиск. Зададимся радиусом $R_0 = \frac{S_M - s}{2}$. Вычислим ошибки поиска для всех блоков, центры которых находятся на расстоянии R_0 от центра макроблока (здесь под расстоянием на дискретной плоскости подразумевается мера $\rho(A,B) = R \Leftrightarrow (|X_A - X_B| = |Y_A - Y_B| = R)$. Выберем блок с наименьшей ошибкой поиска, примем центр этого блока за новый центр поиска, уменьшим радиус в два раза и произведем еще одну итерацию поиска. Итерации прекращаются либо при $R_n = 1$ пикселю, либо по достижении малого порогового значения ошибки поиска, либо когда ошибка поиска для центра очередной итерации меньше ошибок для окрестных блоков. При использовании этого алгоритма перебирается всего лишь $\log_2(S_M - s)$ векторов, что существенно ускоряет поиск. Проблема этого метода в низком качестве найденного вектора движения, ошибка поиска получается больше, чем при использовании вышеприведенных методов.

Поиск с предварительным масштабированием. Этот метод заключается в том, что предварительно блок и макроблок уменьшают в два раза по вертикали и горизонтали каким-нибудь быстрым способом, например, простым усреднением по четырем точкам. Затем в уменьшенных блоках проводится поиск либо полным перебором, либо спиралевидным перебором. Далее найденный вектор (x, y) увеличивается в два раза и проводится его уточнение путем рассмотрения 9-ти окружающих векторов: $(x \cdot 2, y \cdot 2)$,

$(x \cdot 2 \pm 1, y \cdot 2 \pm 1)$. Из них выбирается вектор, дающий минимальную ошибку. Такой способ позволяет достаточно быстро и точно найти вектор движения и поэтому он используется чаще всего.

Поскольку только очень медленный точный перебор дает оптимальный вектор движения, а также в силу того, что в реальном движении объекты состоят не только из квадратных блоков, на практике методы поиска векторов применяются в сочетании с алгоритмами уточнения векторов и улучшения предсказания движения. Рассмотрим некоторые из них.

Нахождение локального минимума ошибки. Суть метода ясна из его названия. После нахождения первого приближения вектора движения проводятся несколько итераций по нахождению локального минимума ошибки поиска в однопиксельной окрестности вектора.

Поточечная интерполяция вектора движения. Поскольку в реальных видеопоследовательностях движение объектов происходит не квадратными блоками, имеет смысл произвести поточечную интерполяцию векторов движения. Пусть найдены вектора для всех блоков текущего кадра $\{P_{n,m}\}$. Интерполируем вектор движения на все точки текущего блока с помощью поверхности второго порядка $P(x, y) = ax^2 + bxy + cy^2 + dx + ey + f$, где x, y — координаты точки внутри блока. Должны быть соблюдены граничные условия: на границах блоков вектора движения равны среднему вектору движения граничащих блоков (для внутренних точек границ это два блока, для угловых точек — четыре блока). Получается система из девяти уравнений на шесть неизвестных. Эта система решается любым удобным способом. Полученные вектора движения для каждой точки кадра учитывают движения не только одного блока, но и соседних с ним. Недостаток этого способа — большая сложность вычислений с плавающей запятой. Следует заметить, что этот способ применяется не только при декодировании. Возможна вариация алгоритма, в которой во время кодирования производится поточечная интерполяция векторов движения, вычисляется ошибка и для каждого блока сохраняется битовый признак — производить или не производить интерполяцию для данного блока.

Компенсация движения перекрывающимися блоками. Этот метод является блочным аналогом предыдущего метода. При декодировании очередного блока учитывается не только его вектор движения, но и вектора четырех соседних блоков. Отложим вектора движения четырех соседних блоков в текущем макроблоке — получим блоки L, R, T, B (левый, правый,

верхний и нижний соседние соответственно). Составим из них две матрицы LR и TB следующим образом: в матрицу LR войдут первые $\frac{n}{2}$ столбцов матрицы L и последние $\frac{n}{2}$ матрицы R, а в матрицу TB войдут первые $\frac{n}{2}$ строк матрицы T и последние $\frac{n}{2}$ строк матрицы B, расположенные в тех строках/столбцах, в которых они располагались в исходных матрицах. Далее, пусть A — текущий блок, полученный по текущему вектору движения, новое значение текущего блока вычисляется по формуле: $C(x,y) = A(x,y) \cdot H_0(x,y) + LR(x,y) \cdot H_1(x,y) + TB(x,y) \cdot H_2(x,y)$, где

$$H_0 = \begin{pmatrix} 4 & 5 & 5 & 5 & 5 & 5 & 5 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 6 & 6 & 6 & 6 & 5 & 5 \\ 5 & 5 & 6 & 6 & 6 & 6 & 5 & 5 \\ 5 & 5 & 6 & 6 & 6 & 6 & 5 & 5 \\ 5 & 5 & 6 & 6 & 6 & 6 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 4 & 5 & 5 & 5 & 5 & 5 & 5 & 4 \end{pmatrix}, \quad H_1 = \begin{pmatrix} 2 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \\ 2 & 2 & 1 & 1 & 1 & 1 & 2 & 2 \\ 2 & 2 & 1 & 1 & 1 & 1 & 2 & 2 \\ 2 & 2 & 1 & 1 & 1 & 1 & 2 & 2 \\ 2 & 2 & 1 & 1 & 1 & 1 & 2 & 2 \\ 2 & 2 & 1 & 1 & 1 & 1 & 2 & 2 \\ 2 & 2 & 1 & 1 & 1 & 1 & 2 & 2 \\ 2 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \end{pmatrix},$$

матрица H_2 получается из матрицы H_1 поворотом на 90°.

Таким образом, данный метод корректирует текущий блок, основываясь на данных о движении соседних блоков. В самом простом варианте метод не требует хранения никаких дополнительных данных. Аналогично вариации предыдущего метода, при кодировании можно хранить битовый признак — проводить или не проводить компенсацию перекрывающимися блоками, основываясь на сравнении ошибки с компенсацией и без компенсации. Добавим, что этот метод достаточно быстро работает и дает неплохие результаты.

4. КОДИРОВАНИЕ ОСТАТОЧНОГО ИЗОБРАЖЕНИЯ

Поскольку изображение, построенное из предыдущего кадра по векторам движения, довольно сильно отличается от кодируемого кадра, необхо-

димо провести дополнительную коррекцию изображения. Для этого вычисляется так называемое остаточное изображение. Оно равно поточечной разности оригинального кадра и его аппроксимации, построенной по векторам движения.

В MPEG-подобных видеокодерах остаточное изображение кодируется аналогично ключевому кадру. Сначала применяется ДКП, затем квантование. Следует заметить, что величины в остаточном изображении имеют меньшие значения, чем в ключевом кадре, следовательно, меньше будут и коэффициенты ДКП. Матрица ДКП для остаточного изображения будет состоять преимущественно из высокочастотных коэффициентов. Поэтому квантование остаточного изображения должно быть более мягким, чтобы обеспечить приемлемое качество изображения.

В кодировании остаточного изображения имеется еще один нюанс. Если вектор движения для какого-то блока был найден плохо, т.е. ошибка поиска велика, то может оказаться выгодней кодировать не вектор движения и остаточное изображение для блока, а собственно исходный блок так, как это делалось в ключевом кадре.

5. ОКОНЧАТЕЛЬНОЕ КОДИРОВАНИЕ ДАННЫХ

Для кодирования получившихся данных — вектора движения, отквантованные коэффициенты и т.д. — обычно применяются энтропийные методы кодирования. Они основаны на составлении таблицы частот вхождения символов в кодируемую последовательность с последующим присвоением наиболее часто встречающимся символам наиболее коротких кодов.

Типичным представителем таких способов кодирования является код Хаффмана. Обычно пользуются его модификацией — динамическим кодом Хаффмана, который не требует хранения таблицы частот, поскольку эта таблица может быть построена во время декодирования.

Еще одним представителем класса энтропийных кодировщиков является арифметический кодер. Скорость его работы немного ниже, чем у кода Хаффмана, но он позволяет достичь более высоких коэффициентов сжатия. Трудность его использования заключается в том, что патенты на большую часть модификаций арифметического кодера принадлежат ряду крупных фирм, тогда как код Хаффмана является свободным для использования.

Код Хаффмана и арифметический кодер являются универсальными кодировщиками, пригодными для данных любых типов. Если же имеется некоторая информация о кодируемых данных, то выгоднее применять специ-

альные методы кодирования или преобразования, которые уменьшат объем данных перед универсальными кодерами. Рассмотрим некоторые подходы такого рода.

Аппроксимация по соседним элементам. Когда кодируется последовательность данных, элементы которой соответствуют блокам кадра (коэффициенты ДКП, компоненты векторов движения и т.д.), то имеет смысл применять такой метод аппроксимации. Элементы, соответствующие первой строке и первому столбцу блоков, остаются неизменными. Все остальные элементы записываются как разность элемента и полусуммы его верхнего и левого соседей. В силу предположения о том, что близко лежащие блоки ведут себя примерно одинаково, такое преобразование позволит уменьшить амплитуду элементов и, следовательно, увеличить коэффициент сжатия.

Квадро-дерево. Этот способ применяется для кодирования двумерных массивов битовых признаков. Суть его заключается в следующей рекурсивной процедуре. Если кодируемый массив состоит из одинаковых символов, то сохраняем специальный код, код этого символа, и прекращаем итерации для этого массива. Если в массиве присутствуют и нули, и единицы, то разбиваем массив на четыре четверти и к каждой из них снова применяем ту же процедуру. Рекурсия прекращается в случае, когда одно из измерений массива становится равным 1.

6. ПРЕ- И ПОСТФИЛЬТРАЦИЯ ВИДЕОПОСЛЕДОВАТЕЛЬНОСТЕЙ

В заключение обзора MPEG-подобных методов видеокодирования скажем несколько слов о предварительной и конечной фильтрации видеопоследовательностей.

Префильтрация. Основной задачей префильтрации является подготовка изображения к сжатию. Она обычно состоит в обработке данных набором высокочастотных фильтров, удаляющих шумы, повышая тем самым коэффициент сжатия. Поскольку кодирование является довольно трудоемкой вычислительной задачей, префильтр должен работать очень быстро. Поэтому традиционные схемы фильтрации не всегда подходят для практического использования. Зачастую применяют простейшие сглаживающие схемы типа «крест», которые могут быть легко оптимизированы под конкретный процессор.

Постфильтрация. В силу того, что на всех этапах кодирования видеопоследовательности каждый кадр был разбит на блоки, которые обрабатывались отдельно, на декодированном изображении зачастую видны границы между блоками. Это так называемый эффект блокинга. Для уменьшения этого эффекта применяются различные схемы деблокинга, суть которых в том, что они сглаживают значения на границах блоков в пределах 1-2–ух точек с каждой стороны.

Еще один вид артефактов, появляющихся при декодировании, это так называемый москито-эффект или эффект Гиббса. Он возникает из-за того, что на этапе квантования коэффициентов ДКП происходит отбрасывание высокочастотных гармоник. Поэтому при обратном преобразовании возникают шумы в высокочастотной области. Визуально они проявляются в виде расплывчатых разводов вокруг четких границ объектов, например, на границах букв. Эту проблему частично решают демоскито-фильтры, которые являются высокочастотными фильтрами, настроенными на определенную частоту шума, которая зависит от коэффициента квантования блока.

СПИСОК ЛИТЕРАТУРЫ

1. **Рабинер П., Гоулд Б.** Теория и применение цифровой обработки сигналов: Пер. с англ. — М.: Мир, 1978.
2. **Прэтт У.** Цифровая обработка изображений. Кн.1 и 2: Пер. с англ. — М.: Мир, 1982.. — 312 и 480 с.
3. **Desmet S., Deknuydt B., Li N., Van Eycken L., and Oosterlink A.** A simple algorithm to extract realistic motion fields out of video sequence // Proc. of the EOS-SPIE Intern. Sympos. on Fiber Optic Networks and Video Communications, Berlin, Germany. — 1993. — Vol. 1977. — P. 248–254.
4. **Wuyts T., Van Eycken L., and Oosterlink A.** Calculating motion vectors for an interpolated motion field: Proc of the Conf on Advanced Image and Video Communications, SPIE, Amsterdam. — 1995. — Vol. 2451. —P. 148–155
5. **ITU-T** Recommendation H.263: Video coding for low bitrates communication. — 1996.

В. Н. Касьянов, Г. П. Несговорова, Т. А. Волянская

ВИРТУАЛЬНЫЙ МУЗЕЙ ИСТОРИИ ИНФОРМАТИКИ В СИБИРИ*

1. ВВЕДЕНИЕ

Современные музеи России, существующие почти в каждом городе и являющиеся в большинстве своем краеведческими, в связи с их недостаточным финансированием не всегда имеют возможность включать в свои экспозиции информацию, связанную с последними достижениями в области науки и техники как в своем регионе, так и по всей стране. Специализированные же музеи для российской глубинки — непозволительная роскошь. Заметим, что крупнейшее в России собрание музейных экспонатов по вычислительной технике собрано лишь в Москве, в Политехническом музее, и не доступно большинству россиян. Поэтому в настоящее время наряду с традиционными музеями широкое развитие получают электронные или виртуальные музеи различной тематики, доступные в среде Интернет.

Информатика сформировалась как наука в середине 50-х годов прошлого столетия и менее чем за полувековой период шагнула далеко вперед. С годами от нас уходят активные участники и свидетели ее первых шагов, многое забывается, становится труднодоступным или безвозвратно утерянным. Постоянное развитие информатики и сверхмощное давление зарубежной вычислительной науки усиливают этот процесс, и нужны целенаправленные действия, чтобы богатый отечественный опыт не забывался и мог быть востребован. Без понимания прошлого трудно двигаться вперед.

Нужно сказать, что исследования по истории информатики в передовых странах мира ведутся достаточно широко. Вместе с тем, до недавнего времени история информатики в Восточной Европе и бывшем Советском Союзе была практически неизвестна на Западе, хотя отдельные работы, посвященные этим вопросам, публиковались [1, 2].

В докладе представлен проект по созданию 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 № 02-05-12010). Си-

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

бирская школа информатики и программирования была третьей по значимости в СССР после школ Москвы и Киева и, несмотря на сегодняшние трудности, переживаемые наукой и образованием в России, продолжает играть свою роль и поныне, более чем через десять лет после смерти ее основателя Андрея Петровича Ершова. Это позволяет самостоятельно исследовать становление и развитие информатики в Сибири, точнее в Новосибирском научном центре, на фоне российского и мирового процессов.

Подавляющее большинство из представленных в настоящее время в сети Интернет виртуальных музеев основано на использовании традиционных гипермедиа-технологий. Одним из ограничений традиционных гипермедиа-систем является то, что они предоставляют одно и то же информационное содержание и один и тот же механизм навигации всем пользователям. Вместе с тем, виртуальный музей, на наш взгляд, предназначен для использования различными категориями пользователей, и посетители музея с различными предпочтениями, целями, знаниями, интересами могут нуждаться в различных частях содержащейся информации и использовать различные пути для навигации. Поэтому при создании музея истории информатики в Сибири мы стараемся уделять особое внимание вопросам адаптации.

Структура доклада следующая. В разд. 2 кратко рассматривается сибирская школа информатики и программирования. Разд. 3 посвящен структуре виртуального музея и его содержимому, а разд. 4 – категориям пользователей. В разд. 5–7 изучаются вопросы адаптивной гипермедиа и описывается пользовательский интерфейс виртуального музея. Разд. 8 – заключение.

2. СИБИРСКАЯ ШКОЛА ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ

Начало работ по программированию и информатике в Сибирском отделении АН СССР относится к моменту приезда в новосибирский Академгородок Алексея Андреевича Ляпунова и его ученика — Андрея Петровича Ершова, всесторонне талантливого представителя первого в советских вузах массового выпуска по специальности "Программирование", в то время заведующего лабораторией автоматизации программирования Вычислительного центра АН СССР. Андрей Петрович Ершов — один из тех ученых, которые росли вместе с Сибирским отделением АН СССР, чья деятельность создавала авторитет и научную известность работам этого отделения.

Созданная в Новосибирске академиком А. П. Ершовым и его учениками авторитетная школа программирования, пользующаяся мировой известностью, внесла значительный вклад в становление и развитие теоретического и системного программирования.

Теория схем программ — одно из наиболее крупных достижений в этой области. На ее базе разработаны методы оптимизирующей трансляции, значительно повышающие эффективность и надежность решения задач на ЭВМ с использованием языков высокого уровня. Внесен существенный вклад в теорию и методологию структурного программирования и параллельной обработки, включая автоматическое распараллеливание программ. Разработаны эффективные алгоритмы анализа, верификации и преобразования программ и систем на базе теоретико-графовых и сетевых моделей. Завершается работа по созданию “энциклопедии” теоретико-графовых алгоритмов для программистов. Получены крупные результаты в разработке теории и методов конструирования качественного программного обеспечения на основе смешанных вычислений, конкретизирующих преобразований и языков спецификаций.

Органическое объединение теоретических исследований с созданием экспериментальных и прикладных программных систем, воплощающих и практически проверяющих разработанные идеи и подходы, — характерная черта таких работ. Эти работы охватывают широкий спектр областей системного программирования: трансляторы и транслирующие системы (АЛЬФА, АЛГИБР, АЛЬФА-6 и др.), языки и системы программирования (Эпсилон, Барс, Лисп, Сетл, Бета и др.), операционные системы и системное наполнение прикладных систем (АИСТ-0, СОФИСТ, ЭКСЕЛЬСИОР и др.), системы анализа и преобразования программ (ТМ, ТРАП, АС, СКАТ, СПЕКТР и др.), инструментальные окружения программирования (СОКРАТ). Особенностью реализованных систем, помимо производственных возможностей, является их принципиальная новизна. Ряд созданных систем закладывал новые направления системного программирования.

Полученные результаты в большой степени формируют уровень отечественных работ по теоретическому и системному программированию и служат базой для продолжающихся в Институте систем информатики им. А.П. Ершова СО РАН исследований в области автоматизации программирования для новых архитектур и современных информационных технологий.

Наибольший общественный отклик получили работы А. П. Ершова в области школьной информатики, впервые анонсированные им в докладе "Откуда берутся люди, способные создавать надежное программное обес-

печение" на международной конференции в Лос-Анджелесе в 1975 г. Он инициировал широкий спектр работ по информатизации образования, в результате которых всего через 10 лет произошло эпохальное для нашей страны событие, осознанное в мире лишь в последние годы, — возник курс “Основы информатики и вычислительной техники”, продвинувший компьютер и науку о нем в среднюю школу. Если в середине 80-х гг. в развитых странах видели необходимость лишь в вузовском преподавании информатики, то в настоящее время есть даже международный стандарт на изучение информатики и программирования в средних учебных заведениях.

В рамках работ по компьютерной грамотности сформулирована "Концепция информатизации образования" и определен "Рабочий план" ее реализации более чем на два десятилетия. Созданы методические пособия для школьного учителя по информатике и школьный учебник, основные идеи которого воспроизводятся в учебниках новых авторских коллективов. Разработаны и массово распространены комплекты учебных программных средств и программное обеспечение для непрофессиональных пользователей на типовых школьных ПЭВМ (Робик, Рапира, Школьница и др.).

В дальнейшем выполнено исследование научно-методических основ преподавания информатики и программирования в рамках общего и специального образования. Разработаны и апробированы методики модульно-вариантного обучения информатике и программированию в рамках многоуровневой системы государственного (школа, колледж, университет) и досугового (летняя, воскресная и заочная школы) образования. Подготовлен комплект программных средств и методических пособий для поддержки общего и специального обучения по информатике и программированию.

3. СТРУКТУРА МУЗЕЯ И ЕГО СОДЕРЖИМОЕ

Создавая виртуальный музей истории информатики в Сибири, мы рассмотрели структуру реальных музеев, чтобы выделить в ней те структурные компоненты, которые хотели бы отразить в нашем виртуальном варианте, и ввести соответствующую терминологию.

Виртуальный музей, по нашему замыслу, включает в основном те же составляющие структурные единицы, которые присущи и реальным музеям. Реализованные на данный момент базы данных (БД) виртуального музея истории информатики предусматривают хранение и обработку информации о следующих объектах: публикациях, документах архива, проектах, ученых-информатиках, коллективах, событиях, конференциях и вычисли-

тельной технике. Все вышеперечисленные объекты являются «экспонатами» виртуального музея. Сведения обо всех экспонатах хранятся в соответствующей БД. Каждый экспонат имеет следующие характеристики: порядковый номер, уникальный универсальный идентификатор (UUID) соответствующего объекта, имя автора, добавившего экспонат, дата добавления экспоната, возможность модификации, возможность участия в выставках и права изменения.

Экспонаты, объединенные по тематическому, хронологическому или типологическому критерию, составляют «экспозицию» или «экскурсию». Каждая экспозиция (и экскурсия) имеет следующие характеристики: UUID, название, имя автора, создавшего экспозицию (экскурсию), ссылку на основной файл экспозиции (экскурсии), краткую аннотацию и правила посещения.

Отличие между экскурсией и экспозицией состоит в следующем. *Экскурсия* — это протекающий во времени рассказ о музее, в ходе которого происходит демонстрация экспонатов музея в определенной последовательности. Экскурсия может иметь, например, форму клипа или презентации для Microsoft PowerPoint и может проводиться для пользователя не только в режиме on-line, но иногда и в режиме off-line. В отличие от экскурсии *экспозиция* предполагает, что экспонаты, составляющие экспозицию, посетитель просматривает сам, причем только в режиме on-line. Обычно экспозиция предоставляет пользователю несколько способов навигации, в том числе возможность свободно перемещаться по экспонатам.

Все имеющиеся в музее экспозиции (и экскурсии) подразделяются на постоянные и временные. Постоянные и временные экспозиции составляют «зал экспозиций», а экскурсии — «зал экскурсий». Оба этих зала являются «открытыми», т.е. доступными для просмотра всеми пользователями музея.

В музее также имеются запасники — залы, доступные только для зарегистрированных пользователей: библиотека, архив, хроника событий, зал ученых-информатиков, зал коллективов, зал проектов, зал вычислительной техники, зал конференций, зал новых поступлений и зал подготовки экспозиций и экскурсий.

В *библиотеке* собраны книги, монографии, сборники статей, учебные и методические пособия, статьи из научных журналов, тезисы конференций и т.д.; каждый экспонат библиотеки снабжен UUID, названием, списком авторов, аннотацией, названием источника и файлом, содержащим полный текст (или ссылкой на него). *Архив* представляет собой совокупность текстовых, графических, звуковых и видео материалов; все документы, хранящиеся в архиве, имеют следующие характеристики: UUID, название, да-

та, краткое описание и файл (или ссылка на него). *Хроника событий* включает описания наиболее выдающихся событий из истории развития информатики в Сибири; каждое событие характеризуется следующими параметрами: UUID, название, дата, статус, краткое и полное описание. *Зал информатиков* содержит информацию о наиболее выдающихся ученых-информатиках, включая биографии, основные печатные труды и достижения, фото и пр.; каждый информатик представлен следующей информацией: UUID, Ф.И.О., данные об образовании, занимаемые должности, данные о защищенных диссертациях, данные об ученых степенях и званиях, научные интересы, текст биографии, фото, основные публикации и проекты. В *зале коллективов* содержатся данные о коллективах: группах, лабораториях и институтах; каждый коллектив обладает следующими характеристиками: UUID, название, адрес, краткое и полное описание. В *зале проектов* размещены данные о проектах, создаваемых в рамках работ по информатике (темы, системы); проекты имеют следующие атрибуты: UUID, название, дата начала и окончания, краткое и полное описание. В *зале вычислительной техники* расположены экспонаты, имеющие отношение к вычислительной технике, которая использовалась и разрабатывалась с начала создания Сибирского отделения Академии наук; каждый экспонат имеет следующие параметры: UUID, название, дата выпуска, имя разработчика, фото, краткое и полное описание. *Зал конференций* содержит информацию о различных научных мероприятиях; каждое мероприятие представлено следующей информацией: UUID, название, дата и место проведения, статус, краткое и полное описание. Новые экспонаты, добавляемые пользователями музея, помещаются в *зал новых поступлений*. В зале *подготовки экспозиций и экскурсий* размещаются экспозиции и экскурсии, создаваемые пользователями музея.

Таким образом, при создании нашего виртуального музея мы моделируем существующие реальные музеи, внося некоторые коррективы, вызванные его электронной формой.

4. КАТЕГОРИИ ПОЛЬЗОВАТЕЛЕЙ

Все пользователи нашего виртуального музея подразделяются на 2 основные категории: *незарегистрированные* пользователи («*посетители*») и *зарегистрированные* пользователи («*специалисты*»), которые различаются по уровню доступа к информационным ресурсам.

Пользователи категории «посетители» имеют доступ не ко всей информации, хранящейся в музее, а только к той ее части, которая открыта для всеобщего доступа (например, в виде экскурсий и экспозиций). При этом все ресурсы доступны только для просмотра и поиска. Далее, среди категории «посетители» выделяются 2 подкатегории в зависимости от уровня знаний предметной области: «новички» и «эксперты». «*Новички*» имеют возможность просмотра экскурсий, а «*эксперты*» — экспозиций и электронной конференции пользователей.

Пользователям, относящимся к категории «специалисты», доступны для просмотра все имеющиеся в музее информационные ресурсы, включая информацию запасников, закрытую для всеобщего доступа; они могут также участвовать в электронной конференции и делать записи в книге отзывов. Все «специалисты» разделяются на две основные группы в зависимости от уровня доступа к ресурсам: группа «*простых специалистов*», работающих только в зале новых поступлений, и группа *«музейных работников».*

В группе «простых специалистов» выделяются «волонтеры», «экскурсоводы» и «экспозиторы». «*Волонтеры*» имеют права на добавление новых экспонатов, при этом экспонат может быть любого типа. «*Экскурсоводы*» могут создавать собственные экскурсии, а «*экспозиторы*» –- выставки. Добавленные или созданные ими объекты сначала находятся в зале новых поступлений, впоследствии администраторы соответствующих ресурсов (например, «главный экскурсовод» или «главный экспозитор») принимают решение об их включении в музей. Волонтеры, экскурсоводы и экспозиторы не имеют прав на редактирование БД музея.

Группу «музейных работников» можно представить в виде иерархической структуры, на самом верху которой находится «*директор*» (или «*главный администратор»*), обладающий полными правами на администрирование всех БД музея, включая БД пользователей музея. На втором уровне иерархии находятся администраторы отдельных ресурсов музея, которые назначаются «директором»: «главный экспозитор», «главный экскурсовод», «главный библиотекарь», «главный архивариус», «главный летописец (хронолог)», «главный биограф», «главный коллективовед», «главный проектант», «главный инженер», «главный секретарь». Администраторы ресурсов имеют полные права на администрирование БД соответствующих типов ресурсов (экспозиций, экскурсий, библиотеки, архива, хронологии событий, информатиков, коллективов, вычислительной техники, научных мероприятий). В их полномочия также входит администрирование специалистов, работающих с БД соответствующих типов ресурсов. Третий уровень иерархической структуры включает «музейных работников», назна-

чаемых администраторами соответствующих типов ресурсов: «библиотекарей», «архивариусов», «хронологов», «биографов», «коллективоведов», «проектантов», «инженеров», «секретарей». Они имеют ограниченные права на изменение БД соответствующих типов ресурсов (экспонатов, экскурсий, экспозиций, библиотеки, архива, хронологии событий, информатиков, коллективов, проектов, вычислительной техники, научных мероприятий).

5. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС

Реализованная на данный момент БД виртуального музея истории информатики предусматривает хранение и обработку данных о следующих объектах: публикации, документы архива, проекты, события, коллективы, ученые-информатики, вычислительная техника, мероприятия. К настоящему времени спроектирован и реализован гипермедиа-интерфейс к БД для информационного наполнения музея: просмотра, поиска, ввода и редактирования данных обо всех вышеперечисленных объектах, а также механизм для их связывания. Реализован интерфейс для регистрации и аутентификации пользователей музея и ведения электронной конференции пользователей. Интерфейс поддерживает следующие основные функции.

1. Регистрация пользователей. В музее предусмотрена регистрация пользователей. Незарегистрированные пользователи («посетители») имеют ограниченный доступ к просмотру ресурсов музея и не имеют прав на ввод и редактирование ресурсов. Зарегистрированные пользователи получают доступ к просмотру всей имеющейся в музее информации («специалисты»), также они имеют возможность получить права для ввода и редактирования информации («музейные работники»).

Процедура регистрации нового пользователя состоит в заполнении соответствующей формы. Форма содержит обязательные (реальное имя пользователя (Ф.И.О), e-mail адрес и пароль для входа в систему) и необязательные (страна, индекс, адрес) для заполнения поля. Если пользователь желает осуществлять ввод информации в музей (добавление экспонатов, создание экскурсий или создание экспозиций), он должен отметить в регистрационной форме пункты, соответствующие желаемым видам деятельности. Логин (имя пользователя для входа в систему) генерируется автоматически по специальному алгоритму и высылается на указанный пользователем e-mail адрес.

Что касается пользователей группы музейных работников, то их регистрация производится «директором» или «администратором» в зависимости от категории.

2. Работа с БД публикаций, архива, проектов, событий, коллективов, информатиков, вычислительной техники, научных мероприятий. Для всех ресурсов БД реализован механизм поиска по нескольким ключевым критериям, есть возможность проведения поиска по образцу с использованием спецсимволов. Реализованы возможности выбора количества выводимых на страницу результатов поиска и выбора способа сортировки найденных результатов. Как результат поиска выводятся краткие сведения об объектах со ссылкой на полную информацию.

Для всех типов ресурсов, имеющихся в музее, реализован интерфейс для ввода и редактирования. Ввод данных осуществляется путем заполнения соответствующих форм в зависимости от типа ресурсов: формы для ввода общих сведений, дополнительных сведений и формы для связывания объектов. Каждому новому добавленному объекту присваивается автоматически генерируемый UUID. В БД экспонатов автоматически вносятся следующие сведения о добавленном объекте: порядковый номер экспоната, имя автора, добавившего объект, дата добавления, возможность модификации, возможность участия в экспозициях и права изменения. Интерфейс для редактирования данных реализован с помощью возможности редактирования данных, отображаемых в соответствующих полях формы.

Реализован интерфейс для обеспечения взаимосвязи вышеперечисленных объектов (механизм связывания объектов) при вводе или редактировании информации. Связывание объектов осуществляется с помощью выбора соответствующих объектов, которые требуется связать с данным объектом, из списка всех возможных объектов (для каждого типа объектов). В результате генерируются информационные связи между объектами, имеющие вид гиперссылок от данного объекта к объектам, с ним связанным.

3. Электронная конференция пользователей. Реализован интерфейс для ведения электронной конференции для взаимодействия пользователей. Незарегистрированные пользователи имеют возможность только просмотра сообщений в конференции, а зарегистрированные могут отправлять сообщения в конференцию. Интерфейс поддерживает все стандартные функции электронных конференций: отправление нового сообщения в конференцию, ответ на существующее сообщение и поиск сообщений по нескольким ключевым критериям.

6. АДАПТИВНАЯ ГИПЕРМЕДИА

Адаптивная гипермедиа (АГ) — альтернативный традиционному подход разработки гипермедиа-систем. Цель АГ состоит в том, чтобы увеличить функциональные возможности гипермедиа, сделав ее индивидуализированной. Под *адаптивными гипермедиа-системами* (АГС) понимаются все гипермедиа-системы, которые отражают некоторые особенности пользователя, такие как предпочтения, знания, интересы, в модели пользователя и применяют эту модель для адаптации к пользователю различных видимых аспектов системы.

АГС в общем случае поддерживают следующие три вида адаптации: к данным пользователя, к рабочим характеристикам и к данным окружения. *Данные пользователя* включают различные характеристики пользователей: знания, цели, подготовка, опыт в гиперпространстве, предпочтения, интересы и индивидуальные особенности пользователя. *Рабочие характеристики* включают данные о взаимодействии пользователя с системами, которые не могут быть сведены к характеристикам пользователя, но все еще могут использоваться для принятия решений адаптации. *Данные окружения* включают все аспекты пользовательского окружения, которые не связаны с пользователями (например, аппаратные средства, программное обеспечение, пропускная способность сети или местонахождение пользователя).

АГС системы обеспечивают *адаптивное представление содержания* (адаптация содержания гипердокументов) и *адаптивную навигационную поддержку* (адаптация структуры гиперссылок). Смысл методов адаптивного представления состоит в том, чтобы адаптировать содержание страницы, к которой обращается отдельный пользователь, к текущему знанию, предпочтениям, интересам, целям и другим характеристикам пользователя. Основные методы адаптивного представления текста — это дополнительные, предварительные и сравнительные объяснения, варианты объяснения и сортировка. Следующие технологии используются для реализации вышеперечисленных методов адаптивного представления текста: условный текст, стрейч-текст, варианты фрагментов и варианты страниц, фреймовая технология.

Смысл методов адаптивной навигационной поддержки состоит в том, чтобы помочь пользователям найти путь в гиперпространстве с помощью адаптации способа представления ссылок к целям, знанию и другим характеристикам индивидуального пользователя. Методы адаптивной навигационной поддержки используются для достижения нескольких целей адапта-

ции: обеспечить глобальное руководство, локальное руководство, глобальную ориентацию, локальную ориентацию и управление индивидуализированными представлениями в информационных пространствах. Для реализации этих методов применяются следующие технологии: полное руководство, сортировка ссылок, сокрытие ссылок, аннотирование ссылок, генерирование ссылок и адаптация карты.

На абстрактном уровне АГС состоит из следующих трех компонентов: модель предметной области, модель пользователя и модель адаптации. *Модель предметной области* (МО) представляет собой описание информационного содержания и структуры ссылок предметной области на концептуальном уровне (используя множество концептов и концептуальных связей, представленных в виде направленного ациклического графа). *Модель пользователя* (МП) представляет предпочтения, знания, цели, историю навигации и возможно другие релевантные аспекты пользователя, информацию о которых система получает в явном виде от пользователя или неявном — посредством отслеживания взаимодействий пользователя с системой. Основной частью МП является представление знаний пользователя предметной области посредством концептов МО (с помощью оверлейной модели). Основой адаптивной функциональности АГС служит *модель адаптации* (МА). Она состоит из правил адаптации, которые формируют связь между МО и МП и определяют представление генерируемой информации.

7. АДАПТАЦИЯ ИНТЕРФЕЙСА

Вышеперечисленные методы и технологии АГ предполагается применить в виртуальном музее истории информатики в Сибири. Музей создается в виде информационно-поисковой, справочной и обучающей адаптивной гипермедиа-системы, доступной в Интернет.

Для адаптивного представления информации в виртуальном музее предполагается использовать такие методы адаптивного представления информации, как дополнительные, предварительные объяснения и сортировку, а также такие методы адаптивной навигационной поддержки, как полное руководство, а также сортировку, сокрытие, аннотирование и генерирование ссылок.

Предполагается, что модель зарегистрированного пользователя будет состоять из модели категорий, модели знаний и модели предпочтений.

Модель категорий поддерживается для всех зарегистрированных пользователей, модель знаний и предпочтений — для всех категорий, кроме

группы работников музея. Модель категорий моделирует права доступа к информационным ресурсам музея. Она реализуется с помощью статической модели стереотипов (набор пар атрибут-значение). В качестве атрибутов модели используются имена типов ресурсов БД, значениями атрибутов являются права для доступа к соответствующему типу ресурсов (просмотр, добавление, изменение и их комбинации). Каждой категории пользователей соответствует одноименный стереотип, характеризующийся определенными значениями атрибутов.

Модель знаний используется для моделирования знаний пользователем предметной области. Предполагается реализовать модель знаний посредством оверлейной модели, основанной на структурной модели предметной области. Структурная модель используется для представления множества предоставляемой музеем информации в виде структуры взаимосвязанных концептов и отношений между ними (ациклического графа). Смысл оверлейной модели состоит в том, чтобы представить знание конкретного пользователя как перекрытие («оверлей») модели предметной области. Оверлейная модель для отдельного пользователя представляет собой табличную структуру, в которой для каждого концепта предметной области определены значения следующих атрибутов: знание концепта (изучен, не изучен), чтение (прочитан, не прочитан), готовность для чтения (готов, не готов). Оверлейная модель является динамической: по мере просмотра пользователем информации она автоматически обновляется.

Модель предпочтений моделирует различные предпочтения пользователей, в частности, способ представления информации (использование только текста, графики, звука, видео и т.д.). Она реализуется с помощью статической стереотипной модели, атрибутами которой являются вышеперечисленные способы представления информации, а значениями — истина или ложь.

8. ЗАКЛЮЧЕНИЕ

В статье представлен проект создания виртуального музея истории информатики в Сибири, работа над которым ведется коллективом сотрудников ИСИ СО РАН, ИМ СО РАН и НГУ при финансовой поддержке РГНФ (02-05-12010). Хотя проект был начат лишь в 2001 г., его основой послужили работы авторов над страницами по истории информатики в Сибири, размещенные ими в рамках системы поддержки гуманитарных исследований SIMICS [3].

Основная цель создания музея — это сохранение историко-культурного наследия, связанного с созданием и развитием информационных ресурсов, являющихся важнейшим национальным богатством, а также обеспечение свободного повсеместного доступа к ним с целью повышения общеобразовательного и культурного уровня широких слоев населения.

Авторы надеются, что новые информационные технологии, разработанные при создании данного музея, будут полезны при создании виртуальных музеев различного профиля.

СПИСОК ЛИТЕРАТУРЫ

1. **Ershov A.P.** A history of computing in the USSR // Datamation. — 1975. — Vol. 21, N 9. — P. 80–88.
2. **Ershov A.P., Shura-Bura M.R.** The early development of programming in the USSR // A History of Computing in the Twentieth Century. — New York: Acad. Press, 1980. — P. 137–196.
3. **Kasyanov V.N.** SIMICS: information system on informatics history // Proc. Intern. Conf. on Educational Uses of Information and Communication Technologies (ICEUT). 16th IFIP World Computer Congress. — Beijing: PHEI, 2000. — P. 168.

П. Б. Кряженков

ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС ИНТЕГРИРОВАННОЙ СРЕДЫ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ SFP*

1. ВВЕДЕНИЕ

Функциональный язык программирования, как правило, — язык высокого уровня, предоставляющий богатый набор средств для работы с функциями, рекурсиями. Язык Sisal — один из представителей функциональных языков, ориентированный на сложные вычислительные задачи. Основная синтаксическая единица языка — это функция. В этом главное отличие любого функционального языка от императивного. Операторы языка, такие как циклы, ветвления, присваивания, являются функциями. Несколько меняется программистское мышление, удобнее и понятнее становится писать вычислительные задачи. Математическая завершенность языка гарантирует, что при одних и тех же входных параметрах функция выдаст одинаковые результаты (если не использовать глобально изменяемых переменных). Это позволяет легко отлаживать программы, искать ошибки, проводить оптимизационные преобразования и, что самое примечательное, распараллеливать вычисления уже на уровне функций. Sisal — один из удачных представителей семейства функциональных языков, существующих на данный момент. Программисту предоставляется богатый набор средств для осуществления сложных вычислений, проведения больших научных расчетов. Здесь приведем лишь краткую описательную характеристику, более подробно о Sisal можно узнать, например, из [8]. Как и все современные функциональные языки, он позволяет просто и понятно осуществлять операции с массивами, потоками, отражающими потоки данных. Отличительная особенность — Sisal изначально ориентирован на системы параллельных вычислений. В синтаксисе языка явно присутствуют конструкции, которые будут вычисляться конкурентно. Более того, функциональность и математическая завершенность позволяют проводить эффективный анализ на зависимость по данным и распараллеливать некоторые вычисления, которые предполагалось вести последовательно. Эти преимущества делают

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

Sisal все более популярным для осуществления сложных вычислений; появились реализации языка на большинстве известных многопроцессорных архитектур. В силу ориентированности на параллельные вычисления, реализации для персональных компьютеров не создавалось. На маломощных однопроцессорных станциях Sisal — это скорее игрушка, а не инструмент для работы. Реальные преимущества можно почувствовать только на многопроцессорных станциях.

Настоящая статья посвящена некоторым вопросам создания интегрированной среды программирования SFP, которая позволит на маломощных персональных компьютерах выполнять такие ресурсомалоемкие задачи, как набор исходных текстов, выполнение тестовых запусков, отладка программы, эмулирование многопоточности и многозадачности, а затем по сети передавать код на многопроцессорную аппаратную платформу для полного счета. Таким образом ресурсы мощной станции будут использоваться в основном для счета, т.е. ресурсоёмких задач. Так достигается эффективность и производительность вычислительных систем.

Данная статья имеет следующую структуру. Разд. 1 — введение, содержит описание поставленной задачи, обосновывает ее актуальность. Далее идет разд. 2, описание пользовательского интерфейса среды SFP. В разд. 3 описывается реализация редактора системы, какие проблемы возникали и как были решены. Разд. 4 кратко описывает процесс трансляции программы на языке Sisal из текстового в промежуточное представление. Описание разбито на три части — лексический, синтаксический и семантический анализы. И разд. 5 — заключение, результаты проделанной работы и планы на будущее.

2. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС СРЕДЫ SFP

Для запуска интегрированной среды достаточно запустить исполняемый файл среды. Система инициализируется из конфигурационного файла, считывая оттуда параметры среды, настраиваемые на конкретного пользователя, такие как шрифт текстового редактора, цвета, используемые при подсветке синтаксических единиц языка и т.п. Первое, что видит пользователь, — главное окно среды с верхним меню, панелью инструментов (tool bar). Внизу находится строка состояния (status bar), где отображаются динамические подсказки (кратко описывающие действия, которые можно произвести, “кликнув” в данном месте на мышку), состояние клавиш caps/num/scroll lock. Здесь не открыто ни одного файла для редактирования,

потому некоторые кнопки панели инструментов недоступны (они затемнены серым цветом, доступные кнопки панели инструментов выведены в полном цвете), такие как сохранение файла, кнопки работы с буфером обмена, печать на принтере. Меню верхнего уровня приложения в данном случае выглядит следующим образом: File (работа с файлами), View (вид), Help (помощь). Подчеркнутые буквы служат для обозначения горячих клавиш, которые в сочетании с нажатием клавиши Alt вызовут активацию соответствующего пункта меню.

В меню File доступны следующие команды: New (создание нового текстового файла), Open (открытие существующего файла), Print Setup…(установки принтера), 1 2 3 4 (четыре пункта на открытие последних 4 редактированных файлов в редакторе), Exit (выход). Команда New создает новый текстовый файл с именем EditorNN (NN — соответствующий порядковый номер) с пустым содержимым и открывает для него окно редактирования. Можно набирать в открывшемся окне текст. Команда Open выводит на экран стандартный блок диалога File/Open для выбора файла, который предполагается редактировать. После того как пользователь выбрал файл, открывается окно редактирования, в котором текст и выводится. Print Setup представляет стандартный сервис библиотеки MFC для настройки параметров принтера. Следующие четыре пункта показывают, какие последние четыре файла пользователь редактировал в среде. По нажатию на имя соответствующего файла он открывается на редактирование, как будто был выбран в процессе диалога File/Open команды меню Open. Пункт меню Exit — выход из системы. Пока не открыто ни одного файла на редактирование, выполняется простой выход из системы, освобождаются системные ресурсы, занятые приложением, и программа завершает свое исполнение.

Меню View содержит следующие пункты: Toolbar (панель инструментов) и Status bar (строка состояния), которые могут быть в состоянии checked (выбрано) или unchecked (не выбрано). Состояние checked означает, что соответствующий элемент пользовательского интерфейса (панель инструментов, строка состояния) будет отображен в окне редактора, состояние unchecked — будет скрыт.

Меню Help на данном этапе состоит из одного пункта — About Editor...(описание...). Выбор данного пункта меню выводит на экран блок диалога с краткой информацией о системе: версия, разработчик, дата создания.

Рассмотрим более подробно другие элементы пользовательского интерфейса: панель инструментов и строку состояния. Недоступные кнопки

панели инструментов (сохранение файла, работа с буфером обмена Windows) не имеют аналогов в командах меню в данный момент. Активные кнопки (New, Open, About) дублируют команды, вызываемые активацией соответствующих пунктов верхнего меню (File/New, File/Open, Help/About Editor). Панель инструментов является плавающей. Это означает, что пользователь может перемещать ее в пределах главного окна среды, “приклеивать” (dock) к краям, а также помещать в “неприклеенном” состоянии в любом месте. При наведении курсора мышки на кнопку из панели инструментов высвечивается быстрая подсказка (tooltip) около курсора и отображается развернутая подсказка в строке состояния, внизу главного окна. Строка состояния отвечает за вывод соответствующего описания команд панели инструментов, некоторой другой справочной информации, а также за отображение состояния клавиш caps/num/scroll lock.

Несколько иным выглядит внешний вид среды (рис. 1) в рабочем состоянии, если открыты файлы на редактирование. Меню верхнего уровня кардинально изменяется, в главном окне среды отображаются окна редактирования файлов (одно или несколько), также может присутствовать панель инструментов и строка состояния главного окна. У каждого окна редактирования есть своя строка состояния, которая содержит информацию о редактируемом в нем тексте: положение курсора (номер строки/колонки), режим ввода (вставка/переписывание символов). Также внутри окон редактирования доступно контекстное меню.

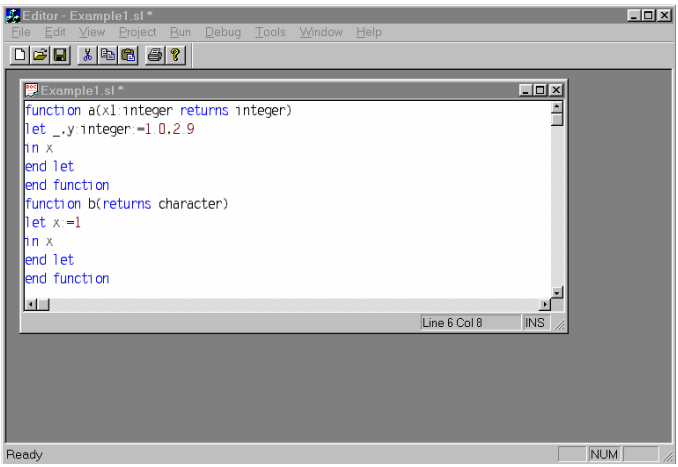


Рис. 1

Меню верхнего уровня включает: File (работа с файлами, выход), Edit (команды редактирования и поиска текста), View (вид), Project (команды управления проектом), Run (запуск), Debug (отладка), Tools (инструменты, настройки), Window (управление окнами редактирования), Help (помощь). Меню View и Help аналогичны меню, описанным в предыдущем пункте. Некоторые меню в системе не реализованы (Project, Debug), они присутствуют в расчете на будущую доработку. Опишем каждое меню более подробно. Некоторые пункты в соответствующих меню также не реализованы (они подсвечены серым цветом и не вызывают никакого действия). Они также присутствуют в системе на будущее.

Меню File состоит из тех же пунктов, которые были описаны в прошлом разделе (New, Open, Print Setup, Exit), с добавлением некоторого количества новых: Close (закрыть), Save (сохранить), Save As…(сохранить под другим именем), Print (печать), Print preview (предварительный просмотр перед печатью). Команда Close вызывает закрытие текущего окна редактирования с проверкой, если файл не был сохранен, то выдается запрос на сохранение файла. Save сохраняет текущий редактируемый файл. Save As… сохраняет текущий редактируемый файл под другим именем, выдается стандартный блок диалога File/Save, где пользователь может выбрать имя записываемого файла, после чего производится запись. Print и Print preview предоставляют интерфейс библиотеки MFC по умолчанию в архитектуре документ/представление.

Меню Edit состоит из следующих подпунктов: Undo (отмена), Cut (вырезать), Copy (копировать), Paste (вставить), Find (найти). Команда Undo отменяет последнее действие, произведенное пользователем в окне редактирования (вставка/удаление символов или блоков текста с клавиатуры или из буфера обмена Windows). Следующие три пункта предназначены для работы с буфером обмена Windows. Cut удаляет выделенный текст из окна редактирования, одновременно копируя его в буфер обмена. Copy просто копирует выделенный фрагмент текста в буфер обмена. Paste вставляет блок текста из буфера обмена в текущей позиции курсора в редактируемом тексте. Команда Find предназначена для поиска фрагмента текста в текущем окне. По этому пункту меню выдается диалоговое окно, где пользователь может задать параметры поиска, такие как сам фрагмент, учет верхнего/нижнего регистра символов, откуда и где искать (сначала текста, от позиции курсора, с конца текста), объем просматриваемого текста (весь или только выделенный), направление поиска (вперед, назад). Результат диалога — новый поиск (Find) или продолжение старого (Find next).

Меню Run содержит пока только один активный пункт — Check syntax (проверить синтаксис). Результат его применения — текущий редактируемый текст пропускается через синтаксический анализатор на предмет выявления синтаксических или семантических ошибок, которые выводятся в соответствующем окне. В случае отсутствия ошибок строится дерево грамматического разбора, которое выдается пользователю в блоке диалога.

Меню Tools состоит из следующих пунктов: IDE Options… (параметры интегрированной среды) и Reread “editor.ini” (перечитать файл конфигурации editor.ini).

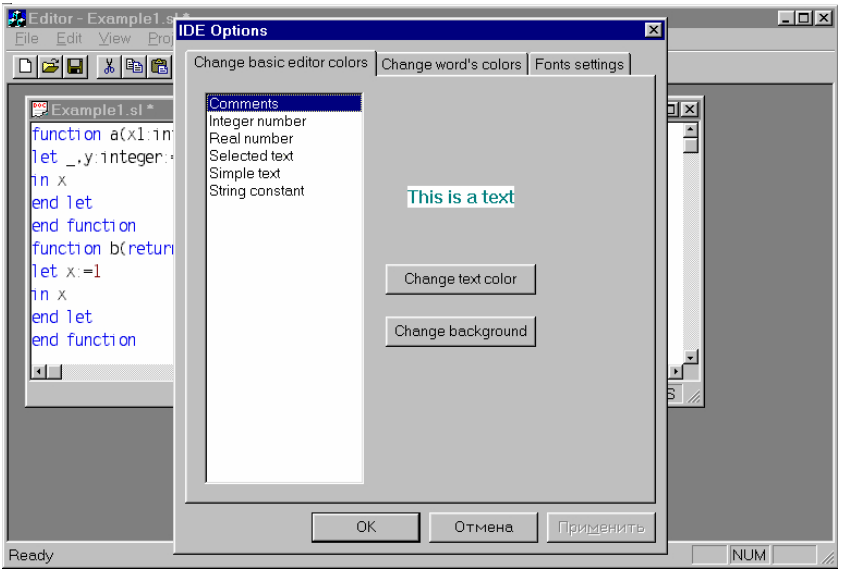


Рис. 2

IDE Options (рис. 2) служит для изменения параметров интегрированной среды программирования. По этой команде на экран выдается блок диалога, состоящий из трех подблоков: Change basic editor color (изменение цветов для синтаксических единиц языка и основных цветов редактора), Change word’s colors (изменение цветов для ключевых слов, рис. 3), Font settings (параметры шрифта редактора). Change basic editor color — в этом блоке диалога можно выбрать, для какого элемента будем менять цвет (комментарии, целые, вещественные или строковые константы, а также выделенный и простой текст), и изменить основной и фоновый цвет.

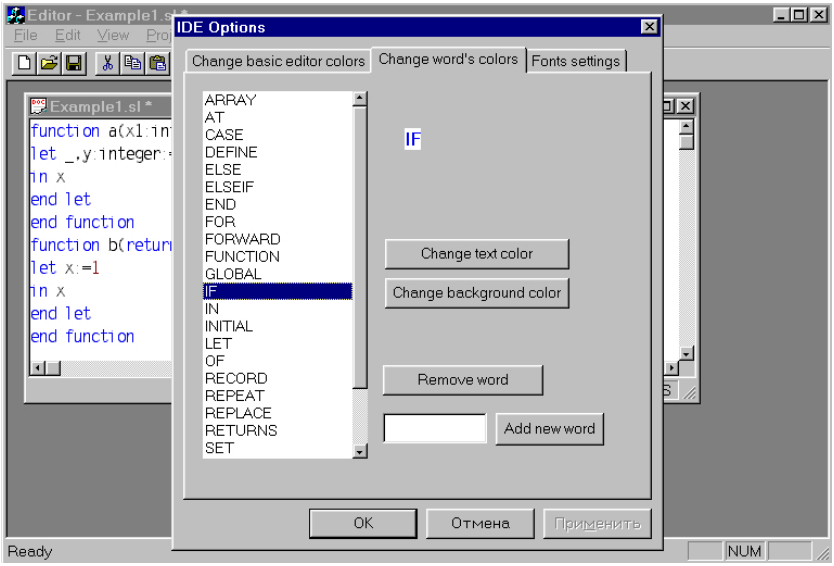


Рис. 3

Change word's colors — здесь можно выбрать одно или несколько ключевых слов и изменить для них цвет фона и тона. Также доступны функции по редактированию набора ключевых слов: добавление и удаление. Font settings — изменение шрифта, используемого в редакторе. Все изменения, сделанные по IDE Options, в случае выхода по клавише "OK" начинают действовать немедленно и сохраняются в конфигурационном файле "editor.ini" (обзор части системы, отвечающей за работу с конфигурационным файлом см. позднее). Reread "editor.ini" перечитывает конфигурационный файл, и все новые настройки начинают действовать немедленно. Данная команда может использоваться для глобального изменения всех настроек без перезапуска системы.

Меню Window отвечает за управления окнами редактирования. Состоит из следующих пунктов: Cascade (каскадом), Tile (мозаикой), 1 2 ... перечислены открытые файлы, в которые можно переключиться для редактирования. Команда Cascade располагает окна редактирования одно над другим, каждое следующее окно чуть смещается от предыдущего по диагонали. Tile располагает окна редактирования в виде мозаики так, что все окна стано-

вятся видны на экране. 1 2 … осуществляет активизацию соответствующего окна и переключает на него фокус ввода. Данные пункты меню позволяют более удобно редактировать текст, если он разбит на несколько файлов.

Панель инструментов состоит из тех же кнопок, только в данном состоянии они все активны. Нажатие на какую-либо кнопку панели инструментов вызывает исполнение определенной команды верхнего меню. Это сделано для удобства пользователя. New соответствует File/New, Open — File/Open, Save — File/Save, Cut — Edit/Cut, Copy — Edit/Copy, Paste — Edit/Paste, Print — File/Print, About — Help/About Editor…

Каждое окно редактирования обладает своей строкой состояния, где отображается информация о текущем редактируемом тексте, а также контекстным меню, которое может быть вызвано нажатием правой клавиши мыши. Контекстное меню содержит команды для ускорения работы пользователя: Copy (копировать), Cut (вырезать), Delete (удалить), Paste (вставить). Delete удаляет выделенный фрагмент текста (как будто нажата клавиша Del), остальные команды работают с буфером обмена Windows: Copy копирует выделенный фрагмент текста в буфер обмена, Cut удаляет из текста и копирует в буфер обмена выделенный фрагмент текста, Paste вставляет блок текста из буфера обмена в текущую позицию курсора.

Интегрированная среда является Windows приложением, а поэтому в нем активно используется мышь. В данном разделе предлагается краткое описание и особенности реализации некоторых функций, которые можно выполнить с помощью мыши. Сюда входят стандартные Windows-функции, как выбор пунктов меню с помощью щелчка на соответствующем названии левой кнопкой мыши, вызов контекстного меню окон редактирования по щелчку правой кнопки мыши, сворачивание/разворачивание и закрытие окон, имеющих системное меню. Используя полосу прокрутки, можно быстро и удобно перемещаться по тексту. Особенно просто, понятно и элегантно можно выделять фрагменты текста с помощью мыши и перемещаться к нужному символу, видимому на экране (для последнего достаточно просто щелкнуть левой кнопкой на символе). В случае выделения большого фрагмента, текст в редакторе автоматически скроллируется в нужном направлении для продолжения выделения.

3. РЕДАКТОР. ОПИСАНИЕ И РЕАЛИЗАЦИЯ

Архитектура документ/представление (document/view) предназначена для упрощения процесса разработки приложения для Windows с помощью

библиотеки MFC. В основе этой архитектуры лежат три глобальных понятия — фрейм, документ и представление. Под документом понимаются те данные, с которыми работает приложение (текст, картинка, ...). Отображение этих данных на экране осуществляется во фрейме документа. В библиотеке для фреймов предусмотрены специальные классы окон — представления, которые отображают данные документа и управляют взаимодействием пользователя с ними. Таким образом, способ хранения данных в памяти или на диске никоим образом не влияет на их внешнее представление пользователю.

В данном редакторе использована архитектура документ/представление. Объект *документ* хранит текст в виде списка строк в памяти, отвечает за операции ввода/вывода (запись и чтение файла). Объект *представление* отвечает за отображение текста на экране пользователя, он хранит информацию, отвечающую за отображение текста, такую как номер текущей строки, колонки курсора, текущий используемый шрифт и т.п. Также на объект представление ложится вся первичная обработка действий, запрошенных пользователем, такие как вставка/удаление текста, перемещения курсора, выделение мышью, работа с буфером обмена Windows.

Редактор интегрированной среды обладает возможностью подсветки текста. Каждой из следующих конструкций можно назначить отдельный цвет: простому тексту, константам (целым, вещественным и строковым) и комментариям. Также цвет можно задать отдельно каждому ключевому слову. Под выражением *задать цвет* понимается задать, каким цветом тона/фона будет изображаться данная конструкция на экране монитора. Цветовые настройки доступны из подблоков диалога в меню Tools/IDE Options. Все изменения в параметрах вступают в силу немедленно.

Реализация подсветки в редакторе достаточно проста: текст раскрашивается построчно. Внутри одной строки сначала выделяются слова, если слово ключевое, то оно закрашивается соответствующим цветом. На следующем шаге проверяется, распознана ли одна из следующих синтаксических конструкций: целая, вещественная или строковая константа. Если нет, то текст изображается как простой текст. После этих шагов, если в строке есть комментарии, то закрашиваются комментарии. И последний шаг, если в строке есть выделенный фрагмент текста, то он окрашивается в нужный цвет. Такой алгоритм отрисовки позволяет перерисовывать автономно только одну строку текста, что можно использовать для оптимизации работы редактора.

Все настройки интегрированной среды (шрифты, параметры цветов) хранятся в конфигурационном файле “editor.ini”. Формат хранения пара-

метров следующий: имя_параметра = значение_параметра. Для текстовых параметров (например ключевое слово) параметры хранятся в явном виде (имя_параметра = текстовое_значение_параметра). Если параметр не является текстовым, то он преобразуется к текстовому виду таким образом, чтобы его можно было восстановить по текстовому представлению. Способ преобразования для каждого параметра свой. Например: C_INTEGER_BK = 255,255,255 означает, что rgb-компоненты цвета фона для целых констант есть соответственно 255,255,255. Регистр символов, используемых при записи параметров, не учитывается, ведущие пробелы игнорируются. В случае переопределения значения параметра (левая часть встречается более одного раза в конфигурационном файле) используется последнее определение.

Опишем кратко реализацию поддержки конфигураций в системе. За все конфигурационные параметры отвечает класс CInitParameters, единственный экземпляр которого создается в объекте CEditorApp (потомок класса CWinApp, стандартный объект приложения в библиотеке классов MFC). В архитектуре документ/представление доступ к экземпляру класса CEditorApp возможен в любом месте. Поэтому экземпляр класса CInitParameters "виден" в любом месте программы и все инициализационные параметры доступны. Данный подход позволяет изменять любые конфигурационные параметры "на лету".

Кратко опишем параметры конфигурационного файла. C_<элемент_подсветки>_Text/Bk — параметр, задающий цвет тона/фона для элемента подсветки. Элемент_подсветки может быть одним из следующих: integer (целая константа), real (вещественная константа), string (строковая константа), comment (комментарий), selected (выделенный текст), <пусто> (обычный текст). C_Word_<ключевое_слово>_Text/Bk — параметр, задающий цвет тона/фона для соответствующего ключевого слова. Значениями описанных выше параметров могут быть тройки десятичных чисел, разделенных запятой: r, g, b, где r, g, b — соответственно значения красной, зеленой и синей компоненты цвета. EditorFont = NNNN… — параметр, определяющий текущий шрифт редактора. NNN… — двоичное представление объекта LOGFONT. Параметр EditorFont вручную менять не рекомендуется, это может привести к непредсказуемым последствиям. Остальные параметры можно изменять в соответствии с приведенными правилами, и хотя для каждого есть визуальная настройка, может понадобиться доводка параметров вручную.

Очень просто производится настройка параметров на конкретного пользователя. После настройки параметров среды надо сохранить конфигураци-

онный файл, и позже в любой момент положить его в каталог системы и дать команду Reread “editor.ini” в меню Tools. Все изменения вступят в силу немедленно**.**

4. ТРАНСЛЯЦИЯ И ПОСТРОЕНИЕ ПРОМЕЖУТОЧНОГО ПРЕДСТАВЛЕНИЯ

Как правило, процесс трансляции состоит из трех основных частей: лексический анализатор, синтаксический анализатор и анализатор семантики. На этапе лексического анализа производится так называемая лексическая свертка программы. Входная программа рассматривается как одна длинная строка, она анализируется посимвольно, и выделяются отдельные лексемы. Представленная таким образом программа подается на вход синтаксического анализатора, задача которого проверить, удовлетворяет ли данная входная строка синтаксису языка. Как правило, одновременно строится промежуточное представление программы для последующего семантического анализа либо семантический анализ может проводиться параллельно с синтаксическим.

Лексический анализ традиционно занимает много машинного времени по сравнению с остальными этапами процесса трансляции. Несмотря на простоту анализа и понятность работы, входная строка (которая может быть достаточно длинной) подвергается посимвольной обработке.

В данной работе применен алгоритм построения непрямого лексического анализатора. Выбранное подмножество синтаксиса языка имеет достаточно простое строение для лексического распознавания. Лексический анализатор интегрирован с синтаксическим анализатором, сгенерированным YACC, потому имеет некоторые характерные особенности, которые будут описаны ниже. Здесь упомянем только то, что при распознавании очередной лексемы возвращается ее код, а само значение лексемы (число для вещественных и целых чисел, строка для идентификатора и т.п.) помещается в специальную переменную. Еще хочется отметить, что специально для YACC существует генератор лексических анализаторов LEXX, однако при разработке среды он не был использован. Как уже отмечалось выше, подмножество синтаксиса языка имеет достаточно простое строение, и потому прямая реализация лексического анализатора эффективнее.

Алгоритм работы реализованного непрямого лексического анализатора следующий:

1. Сначала пытаемся распознать специальные символы, такие как скобки, точки с запятой, запятые, знаки операций, присваивания. В случае успеха возвращается код соответствующего специального символа, никакого специального значения в служебные переменные не помещается.
2. Если это не специальный символ, то значит, либо константа (числовая или строковая), либо идентификатор. Все эти синтаксические конструкции легко определяются по первому символу: если цифра, то это числовая константа; если это буква — идентификатор, если “ — строковая константа; иначе — ошибка. Следующие шаги описывают чуть более подробно, как распознается каждая из этих конструкций.
   2.1. Числовая константа: целая или вещественная. Сначала пытаемся распознать вещественную константу, если удалось, то возвращаем сигнал, что распознана вещественная константа, а значение помещаем в служебную переменную. Если неуспех, то пытаемся распознать целую константу (в этом обязательно будет успех, т.к. первую цифру уже распознали). Возвращаем код, что распознана целая константа, а значение помещаем в служебную переменную.
   2.2. Идентификатор распознается как набор букв, цифр, знака “_”, начинающийся с буквы или _. После распознавания идентификатор заносится (если необходимо) в глобальную таблицу идентификаторов, получается код идентификатора. Результат работы анализатора в этом случае — возвращается сигнал, что распознан идентификатор, а его код помещается в служебную переменную.
   2.3. Строковая константа — все, что встречается во входной строке до следующего знака “, считается строкой. После распознавания строковая константа помещается в таблицу, получается ее код. Возвращается сигнал, что распознана строковая константа, а ее код помещается в служебную переменную.
   2.4. Ошибка. Выдается сигнал ошибки, предпринимаются специальные действия реакции на ошибку (они будут описаны ниже, при рассмотрении синтаксического анализатора).

Как уже говорилось выше, в данной работе при построении синтаксического анализатора использовался инструмент YACC (Yet Another Compiler to Compiler), предназначенный вообще для создания программ, проверяющих корректность входных данных. Полное описание можно посмотреть в Интернете, например [8]. YACC принимает на вход набор грамматических правил, формально описывающих синтаксис входного языка, и генерирует

восходящий LALR(1) синтаксический анализатор в виде подпрограммы на C yyparse(). Построенный синтаксический анализатор обращается к лексическому анализатору, когда ему требуется очередная лексема, который должен написать сам пользователь. Нужная информация по технологическим особенностям YACC будет даваться по ходу изложения того, как построен синтаксический анализатор. Здесь же хотелось бы привести некоторые рассуждения о преимуществах и недостатках построения парсеров с помощью YACC и вручную, а также указать на некоторые проблемы, возникшие при реализации, и пути их решения.

Преимущества заключаются в следующем: спецификацию входного языка достаточно задать в виде набора грамматических правил, что вполне понятно и наглядно; программисту следует мыслить в терминах грамматик, а не в терминах конструкций языка программирования, что значительно снижает вероятность ошибки и позволяет создавать поддержку более крупных синтаксических конструкций. К недостаткам стоит отнести то, что с помощью YACC можно создавать синтаксические анализаторы только для LALR(1) языков (и небольших модификаций). Поддержка обработки ошибок хотя и существует в YACC, однако с гибкостью и легкостью прямых методов программирования здесь сравнивать сложно. Анализаторы, написанные с помощью YACC, достаточно легко расширять поддержкой дополнительных синтаксических конструкций, что при прямых методах делается иногда с трудом.

К проблемам, которые встретились при реализации парсера с помощью YACC, можно отнести некоторую громоздкость получаемых текстов (даже при неполной реализации синтаксиса языка!), отсутствие каких-либо визуальных средств конструирования грамматик. Решение состоит в разбиении на отдельные файлы части текста грамматик, которые отвечают за различные синтаксические конструкции, а для сборки написан небольшой скрипт, собирающий все в один файл. Реализованное подмножество языка Sisal в виде набора грамматических правил, которые подаются на вход YACC, приведено в приложении.

Семантический анализ — следующий этап в процессе трансляции после синтаксического анализа. Подробно об этом можно прочитать в специальной литературе, посвященной построению трансляторов, интерпретаторов и компиляторов, например, [1, 4–6].

Опишем реализацию семантического анализа в данной работе. Проверка семантики языка Sisal осуществляется параллельно синтаксическому анализу, при построении дерева разбора. После того как разобрана очеред-

ная конструкция, построено поддерево, для него вычисляются атрибуты, проверяется корректность использования имен, совпадение типов и, если требуется, автоматическое преобразование типов (например, целый тип расширяется до вещественного; если тип явно не задан пользователем, то задается на данном этапе).

Для арифметических выражений вычисляется атрибут — тип выражения на основе наследуемых поддеревьев. Например, для операции сложения:

a_expr: a_expr + a_expr;

атрибут-тип для конечного выражения вычисляется из типов операндов (если real и integer, то получившееся выражение будет иметь тип real).

К семантическим проверкам, применяемым к арифметическим выражениям, следует отнести проверку корректности использования имен переменных, имен функций (чтобы они были видны в данном контексте), соответствия типов арифметическим (нельзя использовать символьные и логические типы), а также возможность расширения типа выходного выражения (если оно явно задано пользователем). Обязательное требование для вызовов функций — чтобы они возвращали только одно значение арифметического типа.

Рассмотрим семантику оператора цикла for_initial, формальное описание приведено выше. Цикл выполняется, пока истинно логическое условие в while-части оператора. Проверяется, возможно ли использовать переменные и функции в редукциях, операторах присваивания и логических выражениях, а также вычисляются типы возвращаемых оператором значений. Для определения функций также вычисляются типы возвращаемых значений, проверяется, чтобы оператор тела возвращал соответствующие типы, использование имен переменных, аргументов.

Семантический анализ завершает этап процесса трансляции. В случае отсутствия ошибок пользователю выдается сообщение об успехе, иначе неправильные конструкции фиксируются, и в интерактивном режиме предлагается их исправить.

5. ЗАКЛЮЧЕНИЕ

В данной статье рассмотрены вопросы создания компонент интегрированной среды SFP. Кратко описаны внешняя спецификация, пользовательский интерфейс и аспекты реализации следующих компонент интегриро-

ванной среды: редактор, синтаксический анализатор с возможностью интерактивного исправления ошибок. Разработка компонентов интегрированной среды продолжается, в скором времени планируется добавить в список доступных интерпретатор и отладчик с визуальной навигацией по коду. В отладчике должна быть возможность пошагового выполнения программы блоками (например, функциями) до определенной точки останова или до конца программы. Также рассматривается возможность добавления в интегрированную среду преобразователя промежуточного представления для более эффективного исполнения.

СПИСОК ЛИТЕРАТУРЫ

1. **Касьянов В. Н., Поттосин И. В.** Методы построения трансляторов / Отв. ред. А.П. Ершов. — Новосибирск: Наука, 1986. — 344 с.
2. **International** Workshop, selected papers. — Implementation of functional languages. — 1995.
3. **Бирюкова Ю. В.** Sisal 90. Руководство пользователя. — Новосибирск, 2000.
4. **Варсанофьев Д. В., Дымченко А. Г.** Основы компиляции. — Москва. 1991.
5. **Брежнев А. М.** Системное программное обеспечение, трансляторы. — Северодонецк, 1995.
6. **Серебряков В. А.** Лекции по конструированию компиляторов. — Москва, 1993.
7. **Информация** по YACC и созданию трансляторов: http://yacc.chat.ru.
8. **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.

П р и л о ж е н и е

СПЕЦИФИКАЦИЯ ПОДМНОЖЕСТВА ЯЗЫКА SISAL В ВИДЕ НАБОРА ГРАММАТИЧЕСКИХ ПРАВИЛ, ПОНЯТНЫХ YACC

```
%start list

%token NUMBER REAL POWER_OP IDENTIFIER KEYWORD_LET KEY-
        WORD_IN KEYWORD_END
        ASSIGN_OP/*:=*/ KEYWORD_TRUE KEYWORD_FALSE
        NOT_EQUAL LESS_OR_EQUAL GREATER_OR_EQUAL
        KEYWORD_IF KEYWORD_THEN KEYWORD_ELSEIF KEY-
        WORD_ELSE
        KEYWORD_FOR KEYWORD_INITIAL KEYWORD_WHILE KEY-
        WORD_REPEAT
        KEYWORD_RETURNS KEYWORD_OF KEYWORD_ARRAY KEY-
        WORD_SUM KEYWORD_PRODUCT
        KEYWORD_VALUE KEYWORD_LEAST KEYWORD_GREATEST
        KEYWORD_FUNCTION KEYWORD_INTEGER KEY-
        WORD_CHARACTER


%left '-' '+'
%left '*' '/'
%left UNARYMINUS UNARYPLUS /*унарные плюс и минус*/
%right POWER_OP /*операция возведения в степень*/
%left '>' '<' '=' NOT_EQUAL LESS_OR_EQUAL GREATER_OR_EQUAL
%left '|' '&'
%right '~'

%%

list:   /*nothing*/     {;}
        | function_list  | list error;

operator: let_operator | if_operator | for_initial_operator | error;
```

```
/*************************************************************
        описываем функцию
*/
function_list:    function | function_list function;

function:  KEYWORD_FUNCTION IDENTIFIER '(' arguments_list KEY-
      WORD_RETURNS return_type_list ')' operator KEYWORD_END
      KEYWORD_FUNCTION;
/*список аргументов функции*/
arguments_list: /*nothing*/ | arguments_type | arguments_list ';' arguments_type;
/*список переменных(аргументов) одного типа*/
arguments_type:      identifier_list ':' type_declaration;
/*имя типа(декларация имени типа)*/
type_declaration: type_name;
/*имя типа*/
type_name:      reserved_type;
reserved_type:  KEYWORD_INTEGER   | KEYWORD_CHARACTER;
identifier_list: IDENTIFIER | identifier_list ',' IDENTIFIER;
/*список имен типов, возвращаемых функцией*/
return_type_list: type_declaration | return_type_list ',' type_declaration;


 /*************************************************************
  Секция описания операторов
*/


for_initial_operator: KEYWORD_FOR KEYWORD_INITIAL assignment_list
      KEYWORD_WHILE boolean_expr KEYWORD_REPEAT assign-
      ment_list KEYWORD_RETURNS reduction_list KEYWORD_END
      KEYWORD_FOR
      | error;
reduction_list:  reduction;
reduction: reduction_word KEYWORD_OF expr;
reduction_word:     KEYWORD_VALUE | KEYWORD_ARRAY | KEY-
      WORD_SUM | KEYWORD_PRODUCT | KEYWORD_LEAST | KEY-
      WORD_GREATEST;
```

```
let_operator:    KEYWORD_LET assignment_list
            KEYWORD_IN expr_list KEYWORD_END KEYWORD_LET
            | error;
if_operator :    KEYWORD_IF boolean_expr KEYWORD_THEN expr_list
        KEYWORD_ELSE expr_list KEYWORD_END KEYWORD_IF
        | KEYWORD_IF boolean_expr KEYWORD_THEN expr_list elseif_list
        KEYWORD_ELSE expr_list KEYWORD_END KEYWORD_IF
        | error;
elseif_list: KEYWORD_ELSEIF boolean_expr KEYWORD_THEN expr_list
                | KEYWORD_ELSEIF boolean_expr KEYWORD_THEN
        expr_list elseif_list;


/*****************************************************************
            здесь описываются присваивания
*/
assignment_list: assignment | assignment_list ';' assignment;
expr_list:  expr | expr_list ',' expr;
expr:   number_expr | boolean_expr;

/*одиночное присваивание*/
assignment:      variable_declaration_list ASSIGN_OP expr_list | error;
/*объявление списка переменных(типизированных или нет)*/
variable_declaration_list: variable_declaration | variable_declaration_list ',' vari-
        able_declaration;
/*объявление типизированной или нетипизированной переменной*/
variable_declaration: IDENTIFIER ':' type_declaration | IDENTIFIER;

/*****************************************************************
          вызов функции — имя и параметры
*/
function_call:   IDENTIFIER '(' expr_list ')' | IDENTIFIER '(' ')';


/*****************************************************************
          Числовые выражения*/
number_expr: NUMBER_CONSTANT
                | function_call
                | IDENTIFIER
```

```
                | operator
                | '-' number_expr %prec UNARYMINUS
                | '+' number_expr %prec UNARYPLUS
                | number_expr POWER_OP number_expr
                | number_expr '+' number_expr
                | number_expr '-' number_expr
                | number_expr '*' number_expr
                | number_expr '/' number_expr
                | '(' number_expr ')';
NUMBER_CONSTANT: NUMBER | REAL;

/************************************************************
        булевское выражение*/
boolean_expr:   BOOLEAN_CONSTANT
                | function_call
                | number_expr '>' number_expr
                | number_expr '<' number_expr
                | number_expr '=' number_expr
                | number_expr NOT_EQUAL number_expr
                | number_expr LESS_OR_EQUAL number_expr
                | number_expr GREATER_OR_EQUAL number_expr
                | '~' boolean_expr
                | boolean_expr '|' boolean_expr
                | boolean_expr '&' boolean_expr
                | '(' boolean_expr ')';
BOOLEAN_CONSTANT: KEYWORD_TRUE | KEYWORD_FALSE;
%%
```

Ю. В. Малинина

ИС ТРАНСФОРМ: АВТОМАТИЗАЦИЯ НАПОЛНЕНИЯ СИСТЕМЫ*

ВВЕДЕНИЕ

За последние годы число статей, опубликованных в Интернете, значительно увеличилось, но для исследователей по-прежнему актуальна проблема нахождения публикаций, имеющих отношение к конкретной проблеме или области исследования.

Проект по созданию информационной системы по преобразованиям программ продолжается уже нескольких лет. Его целью являлось создание средств для обеспечения исследователей необходимой информацией. В 1999 году была завершена первая версия системы, которая в течение последних лет находилась в опытной эксплуатации. За прошедшее время процесс индексирования в документных системах прошел развитие от ручного заполнения списка ключевых слов до автоматического полнотекстового индексирования внедряемого сегодня и подразумевающего сохранение всех слов текста [10]. Внедрение автоматического индексирования открывает новые возможности перед информационной системой, однако следует учитывать ограничения этого подхода, т. к. согласно последним исследованиям число получаемых при поиске нерелевантных документов подчас достигает 90%, а размер индекса составляет в среднем не менее 40–60% объема документа. С учетом быстрого роста количества электронных документов актуальнось этих проблем усиливается.

Данная статья рассматривает существующие методы индексации и возможные подходы к решению проблемы адекватного автоматического индексирования документов и извлечения из них сопутствующей информации.

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

ОБЗОР ТЕКУЩЕГО СОСТОЯНИЯ СИСТЕМЫ

ИС TRANSFORM предназначена для накопления, систематизации и использования знаний в области преобразования программ, поддержки научных исследований и обеспечения органичной работы с уже сложившимися информационными потоками. Система создана на основе публикаций в области исследования формальных методов оптимизации и преобразования программ. По существу, ТРАНСФОРМ — это соединение хранилища фактов и полнотекстовых источников информации, а также процессов, связанных с обработкой уже имеющихся данных. Если рассматривать указанные части системы: поддержка фактографической информации, возможность работы с полнотекстовыми документами, поддержка процессов обработки данных, то согласно классификации, приведенной в [6], они определяют трехмерное пространство свойств информационной системы. Первая ось (F) характеризует уровень организации хранения фактографической информации, вторая ось (D) — полнотекстовые документы, третье измерение — развитие процессов обработки информации. В рамках этой модели текущее состояние системы находится на начальном уровне и предоставляет следующие возможности.

Первая реализация системы ориентирована на общие описания преобразований программ, однако в дальнейшем собираемые данные предполагается расширить до описания конкретных реализаций преобразований и других аспектов преобразований программ, как то: архитектуры ЭВМ (семейства и модели компьютеров) и языков программирования (оптимизирующих компиляторов). Документальная информация — это библиографические описания публикаций по преобразованиям программ, т.е. авторы, название, источник, издательство, год издания публикации и т.д. Часть публикаций имеет дополнительную информацию: аннотацию, сведения о месте нахождения доступной online-версии, тему публикации и перечень упомянутых в публикации преобразований программ.

БД системы содержит следующие публикации.

- Отечественные публикации с библиографией на русском языке. Часть описаний снабжена аннотацией на русском и/или английском языке. Количество записей — около 572, что составляет более 67,37% от общего объема.

- Зарубежные публикации. Библиографическое описание приводится на языке оригинала и предусматривает перевод заглавия и/или ключевых слов на русский язык. Часть описаний снабжена аннота-

цией на языке оригинала. Количество записей — около 277, что составляет более 32,63% от общего объема.

На сегодня индексировано более 850 публикаций и около 150 преобразований.

К процессам оперирования информацией относятся поиск и ввод данных. Поиск документов осуществляется на основе булевского запроса с логическими выражениями (AND, OR, NOT и т.д.) и возможностью использовать оператор "Like this". Быстрый поиск представляет собой поиск по ключевым словам. Расширенный поиск дополнительно предоставляет возможность настроить более точно параметры поиска по отдельным полям. Поиск с предварительными запросами предоставляет пользователю возможность улучшить формируемый запрос. Все виды поиска разделены также по типу получаемого результата (публикация или преобразование).

Просмотр преобразований можно осуществлять не только при помощи поиска, но и используя дополнительную навигацию в виде рубрикаторов, на которые условно разбивается множество преобразований и публикаций [7].

Текущая версия поддерживает вставку новых библиографических и фактографических данных [1]. На сегодня в системе каждая публикация или преобразование могут быть введены в двух режимах: в пакетном режиме из файла специального формата или через web-интерфейс. Стандарта на обязательные для ввода метаданные на текущий момент не существует, но обычно они включают по крайней мере год публикации, название, автора, краткое содержание — аннотацию и ключевые слова. Стоит отметить, что последние поля (аннотация и ключевые слова) на сегодняшний день заполняются вручную. При этом практически всегда в реальных документах они отсутствуют, поэтому обычно они игнорируются по причине крайне дорогого и медленного их заполнения оператором, вводящим документы в систему. С ведением фактографической информации проблем стало еще больше, т.к. это требует больших усилий по извлечению и предварительной подготовке информации по преобразованиям программ, которые основаны на активном участии специалистов по преобразованиям программ, обладающих достаточной квалификацией.

Следует также отметить, что в текущей версии ИС в базе данных хранится только ссылка на полный текст публикаций, который размещен на некотором сайте. В системе также существует модуль, осуществляющий через определенные промежутки времени доступность этих ссылок и в качестве результата работы при обнаружении недоступной ссылки возвращающий причины, по которым ссылка недоступна. В процессе использования этого подхода была обнаружена следующая проблема: основные свежие

версии научных статей еще слабо организованы и в основном доступны через архивные сайты, сайты научных учреждений, журналов и домашние страницы исследователей, а информация, предоставляемая этими сайтами, постоянно перемещается или уничтожается. Проверка доступности ссылок показала, что 60% ссылок через год после внесения были недоступны и не могли быть впоследствии актуализованы.

ОБЗОР МЕТОДОВ ИНДЕКСИРОВАНИЯ

Прежде чем статьи будут доступны для поиска, их необходимо индексировать. Индекс — это набор слов документа или о документе, по которым этот поиск производится. Основными критериями качества индексирующе-поисковых подсистем являются качество поиска (процент релевантных документов в списке найденных), размер индекса по отношению к размеру документа и скорость поиска по нему.

Рассмотрим в общих чертах проблемы, которые необходимо решить при реализации процесса автоматического индексирования.

Индексирование документа обычно организуется через автоматическую обработку его текста и заполнение метаданных. Автоматическая обработка с полнотекстовым индексированием включает в себя: предварительную конвертацию документа в текстовый формат для публикаций в специальном формате (например, Postscript, Acrobat PDF, TEX или Latex) и преобразование текста документа в набор слов. Причем обычно для слов сохраняется их позиция в документе для обеспечения возможности поиска по словосочетаниям. Существуют два принципиально различных метода такого индексирования с учетом применяемых в дальнейшем методов поиска:

- ***бинарное индексирование*** — не зависит от языка документа по причине бинарной или словарной индексации;

- ***морфологическое индексирование*** — производится с учетом морфологии и семантики языка.

При бинарном индексировании (контекстно-независимом по классификации [8]) поиск ведется на основе алгоритмов “нечеткого поиска”, т.е. поиска с ошибками. В этом случае допускается неполное (с заданным количеством ошибок в начале, середине и конце слова) совпадение слов с шаблоном. Объем индексной информации, полученной из текстовой, может быть в два раза больше, чем исходный текст, и предполагает использование обширного словаря. Поэтому достаточно неэффективно сохранение в базе

данных всего словарного множества документа, предлагаемого в работе [4], при этом, несомненно, нужно отдать должное простоте реализации и быстродействию алгоритма индексирования.

При втором методе индексации (контекстно-зависимом по классификации [8]) слова преобразуются в словоформы с отсечением суффиксов и окончаний, что позволяет искать склонения и спряжения шаблонов. Заметим, что эта же проблема должна решаться и при поиске. Морфологический анализ можно реализовать, оценивая окончания слов в документе [3]. Но если принять, что морфология слов русского языка определяется по окончанию и суффиксу, остается достаточно много слов, выпадающих из этого правила, т.е. есть слова, которые имеют окончание, подходящее для некоторой формы слова, но являются совершенно другой формой. Например, окончание "-ать" указывает на то, что слово является глаголом (прыгать, бежать). Но слово "кровать", оканчивающееся на "-ать", является существительным. Значит, из правил морфологического разбора могут быть исключения. Также есть слова, которые не изменяют свою форму, например: предлоги, наречия, и т.д. Значит, есть дополнения к правилу морфологического разбора. Для неоднозначно трактуемых слов можно использовать специальную таблицу в БД с атрибутами *слово* и *часть речи*, и при анализе просматривать сначала ее, а затем (если слова там нет) выполнять оценку по окончанию слова. В этой же таблице будет находиться и незначительное число слов, принадлежащих неизменяемым частям речи, таким как междометие, наречие и т.п. Аналогичный подход применим и к англоязычным текстам [11].

Заметим, что несмотря на несомненные плюсы, полнотекстовое индексирование в любом своем виде имеет и ряд существенных минусов, т.к. дело не только в том, что в базу данных попадут одинаковые слова, имеющие разные падежные окончания и т.п., но и в более глубокой проблеме, включающей следующие моменты:

- ***много излишней информации в индексе***, т.е. слов, никак не характеризующих документ, а связывающих "ключевые" слова. Это может привести к большому числу нерелевантных документов, которые будут выданы при поиске, если шаблон попадет на " ***излишнюю информацию*** ": глаголы, служебные слова, местоимения и т.д.

- ***большой объем индекса*** за счет " ***излишней информации***" — следовательно, увеличение пространства необходимого для хранения индекса и увеличение времени поиска.

Как указывается в [8], эти недостатки обусловлены самой концепцией такого индексирования. Действительно, с одной стороны, наличие в индексе всех слов текста гарантирует его нахождение по любому из них, но с другой, может существенно затруднить поиск, если текст содержит некоторые лирические отступления, напрямую не связанные с рассматриваемой темой.

Таким образом, мы возвращаемся к выделению "ключевых" слов из документа, для того чтобы гарантировать валидность результатов поиска. Только в отличие от систем первого поколения, в которых применялось ручное индексирование, данный процесс должен выполняться полностью автоматически в связи со значительно возросшим потоком документов. Кроме того, индексирование "ключевых" слов позволит значительно сократить объем индекса, а значит, и время поиска по нему.

"Ключевые" слова — это слова, определяющие содержание документа, характеризующие его смысл. Согласно [8] все многообразие документов можно разделить на виды с точки зрения их организации:

- ***структурированные документы*** — имеют четкую (известную) организацию содержания информации в документе. Определенные поля данных, их последовательность и положение, например: договора, акты, служебные записки и т.д.;

- ***неструктурированные документы*** — не обладают структурой в разрезе полей данных, например: статьи, книги и т.д.

Первый вид — это хорошо структурированные документы, с обработкой которых нет проблем, поэтому данная группа документов в дальнейшем рассматриваться не будет.

Далее разделим неструктурированные документы на подвиды с точки зрения возможности выделения "ключевых" слов. В качестве предпосылки предполагается исходить из того, что метод определения характерных слов документа должен зависеть от того, что в этом документе важно для контекстного поиска. Таким образом, разделим все неструктурированные документы на следующие группы (подвиды):

- ***контекстно-идентифицируемые*** — описывают конкретные вопросы (статьи, заметки, книги и т.д. на определенную тему или по определенным вопросам);

- ***контекстно-неидентифицируемые*** — не несут информации по конкретным вопросам (например, большинство художественной литературы).

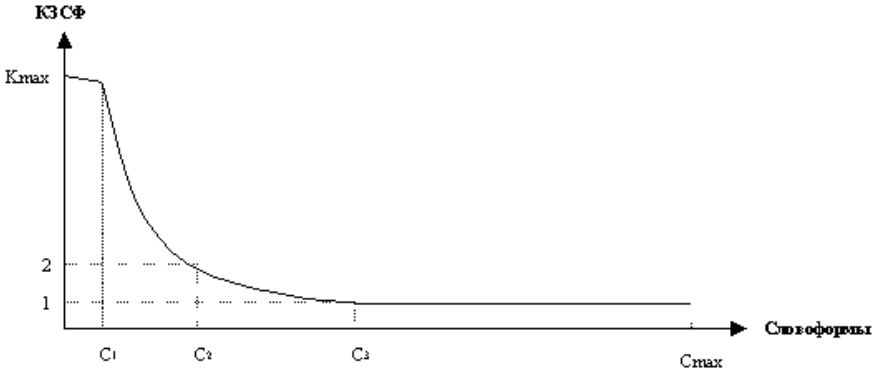
Нас интересует в основном первая группа документов. Она характеризуется тем, что в ней существует явно выделенная тема, о которой идет речь в тексте. Причем описание производится с помощью специальных терминов данной темы и сопроводительных слов, их поясняющих. Анализируя задачи, цели и способы поиска таких документов, можно заметить, что он происходит именно на основе этих самых терминов, которые и будут в данном случае "ключевыми" словами текста.

С учетом вышесказанного рассмотрим две концепции, которые позволяют улучшить условную схему индексирования документа, но предполагают различный подход в выделении ключевых слов.

Индексирование согласно коэффициенту значимости словоформ (КЗСФ)

Согласно подходу, предложенному в [2], процесс автоматического индексирования выглядит следующим образом. Входной документ преобразовывается в поток слов, из которых выделяются словоформы путем отсечения окончаний и суффиксов, при этом для каждой словоформы необходимо запоминать частоту повторений, которую можно рассматривать как весовой коэффициент, отражающий его значимость. Полученный список сортируется по убыванию коэффициента КЗСФ.

Результаты данной работы представлены на графике.



Обозначения:

- C_{max} — число словоформ в тексте документа;
- K_{max} — максимальный КЗСФ;
- C_1 — число словоформ с КЗСФ примерно равным Kmax;
- C_2 — число словоформ с КЗСФ>2;
- C_3 — число словоформ с КЗСФ>1.

Числовые соотношения:

- $K_{max} \approx$ 8–15 для одностраничного документа (A4);
  $\approx$ 50–300 для 5–10 страничного документа (A4);
- $C_1 \approx$ 1–5 в зависимости от документа;
- $C_2 \approx$ 20–30% от Cmax;
- $C_3 \approx$ 50% от Cmax;

Представленные результаты интерпретируются автором, следующим образом.

1. Все слова со словоформами, находящимися правее точки C_3, не должны попадать в индекс, т.к. они не только никак не характеризуют документ, но зачастую к нему не относятся. Причем чем больше документ, тем это корректнее.

2. Слова со словоформами из зоны C_2–C_3 можно отнести к “псевдоключевым” только для очень небольших документов (порядка 1–2 страниц). Для больших документов их лучше игнорировать.

3. Слова из начала списка, покрывающие накопительной суммой своих КЗСФ примерно 5–10% от суммы КЗСФ всех слов списка, позволяют однозначно классифицировать документ по теме в случае, если каждой теме будет сопоставлен список характерных слов.

Таким образом, описанный подход выделения “ключевых” слов может быть реализован следующим образом:

– преобразование полученного текстового документа в поток слов;
– преобразование слов в словоформы;
– создание списка словоформ документа, упорядоченного по КЗСФ;

– выбор точки игнорирования по КЗСФ (ТИкзсф) в зависимости от Cmax;
– занесение в индекс словоформ, расположенных левее Тикзсф.

Он позволяет сократить объем индекса по крайней мере в два-четыре раза в зависимости от выбора точки игнорирования для КЗСФ без потери качества поиска по сравнению с полнотекстовым индексированием. Более того, число выдаваемых системой найденных нерелевантных документов значительно сократится за счет очистки индекса от мусора, а время поиска сократится за счет уменьшения индекса.

Индексирование согласно "предметному указателю"

В [9] предлагается концепция формирования индекса по принципу формирования предметного указателя. Известно, что предметный указатель эффективно используется для поиска в научно-технической литературе. Предметный указатель — это терминологическая база данного текста. Она включает базовые термины (существительные) и уточненные термины (существительные с определяющими их прилагательными и, возможно, предлогами).

Сама структура такого индекса должна обеспечить не только быстрый, но и релевантный поиск. Для повышения релевантности используется распространенный подход: при формировании терминологической словарной базы конкретного документа сохраняется не только сам термин, но и частота его вхождения в документ. Поэтому при выполнении поиска можно упорядочить его результаты по частоте вхождения искомого термина в документ. Кроме того, можно ввести некоторое пороговое значение f (например, $f > 1$), которое должно использоваться в качестве критерия отбора записей в поисковом запросе. Для формирования терминологического индекса требуется решить следующие задачи.

- Определить часть речи слова в документе (морфологический анализ).
- Выяснить, что является составным термином (синтаксический анализ). Предполагается, что простым термином является существительное. С составным же термином дело обстоит сложнее, поскольку нужен достоверный критерий того, какая последовательность слов является терминологически связанной.
- Определить словоформу.
- Удалить все записи с частотой вхождения ниже некоторого порогового значения. Т.е. предполагается, что термины, которые встре-

тились в документе, скажем, один раз, неадекватно характеризуют его содержание. Пороговое значение предлагается подбирать эмпирически.

Естественно, что при данном подходе не на любые запросы будет получен ответ, например, если образец поиска будет содержать только исключенные из терминологического индекса слова. Но, с другой стороны, поиск документов по словам: “например” или “следовательно” не несет особого смысла. Кроме того, учет таких слов может привести к ошибочному выполнению запроса. В качестве примера рассмотрим поиск по словам “можно” и “термин”. Ясно, что если поиск ведется по вхождению в документ хотя бы одного из двух терминов, то возможно, что он весь будет состоять из документов, содержащих слово “можно” (причем с достаточно высокой частотой вхождения) и не содержащих слово “термин”. Если же критерий поиска построен на вхождении в документ обоих слов, то релевантность такого поиска может вызвать сомнения. Не будет удивительным, если частота вхождения слова “можно” значительно превысит частоту вхождения слова “термин”, в результате чего наверху списка окажутся документы, имеющие меньшую релевантность относительно слова “термин”.

Следует отметить критичность данного алгоритма по отношению к точности определения части речи и правильности исключения "незначимых" слов.

В первую очередь сложность вызывает определение критерия, который позволил бы отличить существительное от прилагательного. Причем дело здесь не только в том, что существительное и прилагательное могут иметь в предложении одинаковые окончания, т.е. морфологический анализ в этом случае не сможет нам помочь, но и в том, что существительное и прилагательное могут быть представлены одним и тем же словом. Так слово “данные” в термине “экспериментальные данные” является существительным, а в словосочетании “данные нам в ощущениях” — прилагательным. Для четкой идентификации части речи потребуется достаточно сложный синтаксический анализ.

ЗАКЛЮЧЕНИЕ

Рассмотренные подходы дают существенные улучшения индекса, однако эмпирические подборы пороговых значений являются узкими местами обоих концепций. Недостатком постоянного порогового значения является то, что в небольших документах может не оказаться терминов с частотами,

выше порогового значения, но нельзя сказать, что такие документы вообще не несут никакой информации. Использование переменного порогового значения, принимающего некоторое значение в интервале между максимальной и минимальной частотой терминов в данном документе, было бы предпочтительней, но для этого требуется разработка алгоритма его вычисления. Кроме того, можно определить пороговое значение для совокупности документов [8].

Предполагается комбинация и адаптация этих подходов в новой версии ИС ТРАНСФОРМ, для того чтобы пользователи системы имели доступ к актуальной и релевантной информации .

СПИСОК ЛИТЕРАТУРЫ

1. **Волянская Т., Малинина Ю.В.** Трансформ: интерфейс для ввода информации // Поддержка супервычислений и интернет-ориентированные технологии. — Новосибирск: ИСИ, 2001. — С. 125–139.
2. **Гацко А**. Концепция индексирования по ключевым словам. — computerclub.dore.ru
3. **Ермолаев Д.С.** Компьютерный морфологический разбор слов русского языка. — www.icreator.ru
4. **Игумнов Е**. Основные концепции и подходы при создании контекстно-поисковых систем на основе реляционных баз данных. — www.citforum.ru
5. **Корнеев В.В., Гареев А.Ф., Васютин С.В., Райх В.В**. Базы данных. Интеллектуальная обработка информации. — М.: "Нолидж", 2000. — 352 с.
6. **Красилов Н., Косякин И., Черных Д.** Об одной модели документооборота // Открытые системы. — 1997. — № 1. — www.osp.ru
7. **Малинина Ю.В.** ИС ТРАНСФОРМ: Прототип интерфейса для визуального исследования БД // Проблемы систем информатики и программирования. — Новосибирск: ИСИ, 1999
8. **Марков А.** Концепция построения электронного архива // Открытые системы. — 1997. — №1. — эл. публик. www.osp.ru
9. **Моисеенко В, Майстренко А.** Релевантность полнотекстового поиска: подход на основе построения терминологической базы документов. — www.citforum.ru
10. **Lawrence S., Bollacker K., Lee C.** Indexing and Retrieval of Scientific Literature // Proc. 8th Intern. Conf. on Information and Knowledge Management. — CIKM 1999.
11. **Porter M.F.,** An algorithm for suffix stripping // Readings in Information Retrieval. — 1997. — Vol. 14, N 3. — P. 130–137.

В. А. Маркин, С. А. Маркина

ПРОЕКТ СИСТЕМЫ ДЛЯ БЫСТРОГО ПРОТОТИПИРОВАНИЯ РАСПАРАЛЛЕЛИВАЮЩЕГО КОМПИЛЯТОРА. УНИВЕРСАЛЬНОЕ ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ СИСТЕМЫ*

1. ВВЕДЕНИЕ

Развитие ЭВМ с параллельными архитектурами и высокопроизводительных вычислительных систем ставит перед системными программистами задачи по созданию новых технологических подходов и их эффективному использованию. Одним из направлений является создание языков программирования с явно определенными параллельными семантическими конструкциями, что, впрочем, не облегчает процесс написания параллельных программ. Процесс человеческого мышления ближе подходит к последовательной модели вычислений, чем можно объяснить, почему до сих пор разрабатывается большое количество последовательных алгоритмов любого уровня сложности и в различных прикладных областях. Поэтому вторым основным направлением по созданию программных систем для параллельных архитектур является разработка распараллеливающих и оптимизирующих компиляторов.

В отличие от явного параллелизма, связанного с расширением существующих языков средствами управления параллельными вычислениями, автоматическое распараллеливание программ свободно от многих недостатков первых систем, но имеет свои собственные, связанные с ориентированностью на определенную «среднюю» программу. Различные распараллеливающие системы включают в себя различные наборы оптимизирующих и реструктурирующих преобразований и средства распараллеливания вычислений. Каждая система определяет свой класс входных программ, в которых она может выявить параллелизм. Также не последнюю роль играют конечные параллельные архитектуры, на которые ориентированы распараллеливающие системы [3].

Описываемая в данной работе система для быстрого прототипирования распараллеливающего компилятора выполняется в рамках проекта ПРО-

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

ГРЕСС [1, 2], проводимого в лаборатории конструирования и оптимизации программ ИСИ СО РАН. Началом работ по созданию этой системы для авторов послужили уже полученные программные наработки в области исследования свойств программ в рамках проекта ПРОГРЕСС (в частности, по анализу программных зависимостей; исследованию различных теоретико-графовых внутренних представлений программ и их эффективности для оптимизирующих преобразований). Одной из целей проекта ПРОГРЕСС является исследование свойств различных классов языков программирования, позволяющее выявлять внутренний параллелизм входных программ. В связи с этим накапливается большая база программных библиотек, решающих задачи теории построения трансляторов для высокопроизводительных архитектур, которые пока трудно совместимы между собой, что значительно усложняет их совместное использование. Данная проблема и требует создания подобной системы.

Далее в разд. 2 описывается проект системы по созданию прототипа распараллеливающего компилятора. Даются формальные определения составных частей системы, ее наполнение и содержание. В разд. 3 приводится описание расширяемого универсального внутреннего представления, которое является стержнем всей системы. Определяются структуры внутреннего представления, описываются его компоненты и их взаимосвязь. В заключении приведен перечень проделанной работы и представлены планы на будущее.

2. ОПИСАНИЕ ПРОЕКТА СИСТЕМЫ

Работа с новыми распараллеливающими программными комплексами требует определенных знаний и навыков. Следовательно, возникает необходимость в создании различных программных систем для обучения параллельному программированию. К тому же при построении эффективных распараллеливающих компиляторов для конкретной целевой архитектуры необходимо определять оптимальное множество средств анализа свойств программ, множество оптимизирующих преобразований и т.п. В теории распараллеливания и оптимизации постоянно возникают новые подходы и алгоритмы, анализ которых необходимо проводить в контексте уже существующих, что позволяет определить их оптимальность и эффективность.

Данный проект направлен на создание системы для построения прототипа распараллеливающего компилятора, а также изучения свойств про-

грамм, исследования алгоритмов распараллеливания и оптимизации. Предполагается, что система будет содержать:

1) инструменты для быстрого прототипирования распараллеливающих компиляторов и оптимизирующих компиляторов для различных целевых архитектур, таких как VLIW [14], суперскалярных и мультипроцессорных с разделенной памятью (NUMA [15] и др.). Система должна позволять из имеющегося набора алгоритмов создавать, исследовать и исполнять прототип компилятора, функциональная наполняемость которого будет задаваться пользователем;
2) средства и инструменты для унификации программирования различных подходов и алгоритмов, в частности, универсальное расширяемое внутреннее представление и библиотеку работы с ним;
3) инструменты для подключения к системе новых алгоритмов и преобразований; для проведения исследований эффективных форм промежуточных представлений программ, оптимизирующих и реструктурирующих преобразований (оптимальных условий и стратегий их использования для различных классов программ и вычислительных систем);
4) средства визуального проектирования и проведения обучения студентов методам программирования и оптимизации для параллельных архитектур.

Для достижения данных целей система создается как инструмент для манипулирования программами. Это означает пошаговое преобразование программ, начиная с исходного текста программы на одном из языков высокого уровня (например, C, Паскаль), и далее, пропуская программу через различные блоки, получаем текстовое представление распараллеленной программы либо исполняемый код для запуска программы на одной из целевых архитектур. Центральным понятием работы системы является СЦЕНАРИЙ прототипа создаваемого компилятора. В сценарии определяются множество и порядок исполнения компонент системы, которые делятся на два вида:

- *функциональные компоненты* — реализованные алгоритмы, связанные с преобразованием внутреннего представления;
- *инструментальные компоненты* — интерактивные компоненты, ориентированные на взаимодействие с пользователем, внешние графические интерфейсы.

Однотипные функциональные компоненты объединяются в *функциональные блоки*, такие как блок синтаксического разбора (перевод во внутреннее представление), блок внутреннего представления программ, блок

анализа программных зависимостей, блок оптимизирующих и распараллеливающих преобразований, блок кодогенерации. К инструментальным компонентам относятся: текстовый редактор, графический редактор, редактор внешних связей и т.д. Функциональные и инструментальные компоненты реализуются как внешние библиотеки (dll либо элементы ActiveX) с учетом заданных интерфейсов взаимодействия с системой. Регистрация либо загрузка соответствующих компонент происходит во время запуска системы либо во время начала работы сценария.

Для задания сценария в систему вводится *язык описания сценариев* и, соответственно, интерпретатор для данного языка. Основными объектами языка сценариев будут служить библиотеки, вложенные в них запрограммированые алгоритмы и методы, а также типы объектов внутреннего представления системы. Весь сценарий разбивается на участки исполнения, на каждом из которых определены множество подключаемых компонент системы, последовательность и условия их вызовов. Языку сценариев дается одна из основных ролей в разрабатываемой системе. По сути дела, создание прототипа компилятора будет заключаться в составлении сценария системы и его дальнейшего интерпретирования. На данный момент механизм сценария находится на стадии проработки.

С учетом 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 и исследовать прототип распараллеливающего компилятора.

Описание современных подходов к построению распараллеливающих компиляторов можно найти в [3]. Также общую теорию по созданию трансляторных систем и подходы к их реализации можно найти в [4–6]. Смотри также [11, 12].

Система условно разбивает процесс трансляции на следующие этапы: разбор входной программы и перевод во внутреннее представление (front-end); оптимизирующие и реструктурирующие преобразования (трансформация); кодогенерация.

Для реализации каждого этапа необходимо разработать следующие компоненты.

Для этапа front-end:

- расширяемое универсальное внутреннее представление (ВП), а также библиотеку для работы с ним;

- разбор командной строки и настройку среды, библиотеку работы с конфигурационными файлами;
- библиотека парсеров для перевода текста входной программы во ВП. Данная библиотека необходима для создания многоязыковой системы. На текущий момент предполагается реализация поддержки программ на языках С и Паскаль, в дальнейшем множество языков может быть расширено путем написания дополнительных библиотек и подключения их к системе. Такой же принцип расширяемости будет использоваться и для других компонент;
- библиотека алгоритмов анализа программных зависимостей [3, 16]. К настоящему времени реализованы несколько алгоритмов анализа зависимостей по данным (НОД-тест, тест Банержи, модифицированный алгоритм Шостака [7]) и алгоритм построения подграфа зависимостей по управлению. В данную библиотеку также будут включены алгоритмы построения SSA-формы и теоретико-графовых промежуточных представлений программ (ГПЗ, ИГЗ, идеограф).

Для этапа трансформации:

- библиотека оптимизирующих преобразований;
- библиотека реструктурирующих (в т.ч. и распараллеливающих) преобразований.

Для этапа кодогенерации:

- библиотека кодогенераторов с целью получения объектного кода для различных целевых архитектур.

В систему предполагается включить множество программных библиотек и интерфейсы взаимодействия с системой, используя которые, пользователь сможет создать новые библиотеки алгоритмов и подключить их к системе. Система будет иметь дружественный и понятный интерфейс, что позволит проводить образовательный процесс по вопросам теории трансляции программ и их распараллеливания.

3. РАСШИРЯЕМОЕ УНИВЕРСАЛЬНОЕ ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ СИСТЕМЫ

Основной компонентой системы является расширяемое универсальное внутреннее представление программы, которое служит для связи между блоками системы. Главными целями при его проектировании являлись универсальность (общность), расширяемость и гибкость. Данное представление должно было позволить переводить во внутреннее отображение многие ши-

роко распространенные языки программирования, а также обобщить различные теоретико-графовые внутренние представления, разработанные за последнее время [8–10, 13].

Модель разработанного внутреннего представления можно определить следующей схемой, как показано на рис. 1.

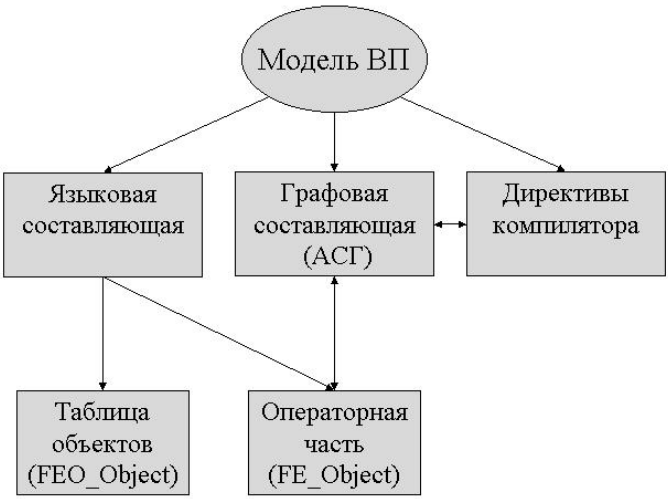


Рис. 1

Языковая составляющая ВП — это множество синтаксических и семантических объектов, иерархически упорядоченных, отображающих большинство общих конструкций входных языков. В качестве базового класса для всех объектов ВП системы используется абстрактный класс OM_object, на базе которого реализован механизм для получения информации об объекте в режиме исполнения (RTTI). Внутренняя реализация данного класса позволяет вести регистрацию всех наследуемых от него типов, динамического создания объектов и контроль типов (рис. 2).

Все объекты распределяются на области видимости, тем самым ВП включает в себя таблицы объектов для каждой области видимости и операторную часть.

Таблицы объектов включают в себя множество определений типов и внутренних объектов. Типы делятся на простые (такие как целый, вещест-

венный, символьный), составные (такие как массив, возможно, динамический, структура) и указатель. Внутренние объекты — это константы, метки и переменные (см. FEO_object иерархию, рис. 2).

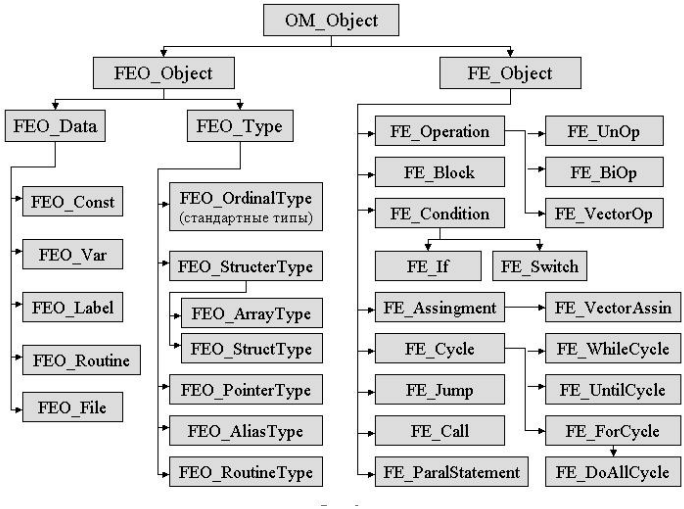


Рис. 2

Операторная часть представлена в виде абстрактного синтаксического графа (который является частью графовой составляющей), узлы которого делятся на семантические и управляющие вершины (см. FE_object иерархию, рис.2). К семантическим вершинам относятся if-вершины, loop-вершины, или операторные вершины. К управляющим относятся вершины-директивы компилятора, или дополнительные вершины ВП, такие как R-вершины графа программных зависимостей. Семантические вершины делятся на вершины высокого и низкого уровней. Вершины операторов низкого уровня можно сравнить с командами модели какой-либо вычислительной машины, на последовательность которых можно разбить операторы вершин высокого уровня и проводить какие-либо оптимизирующие преобразования, такие как вычисление подвыражений, протягивание констант и т.д. Множество конструкций, соответствующих большинству конструкций императивных языков программирования, расширено операторами с параллельной семантикой исполнения.

Любой объект ВП содержит атрибуты, необходимые для визуализации программы. Для каждого объекта хранятся координаты его текстового про-

образа — это номер строки и номер литеры в строке по текстовому файлу для начальной и конечной литер конструкции в тексте исходной программы. К атрибутам визуализации также можно отнести идентификатор исходной программы; вид объекта, цикла, безусловного перехода; информацию о переименовании 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.

Поскольку одной из задач при проектировании системы являлось обобщение и реализация различных теоретико-графовых представлений, была выделена отдельно графовая составляющая ВП — абстрактный синтаксический граф (АСГ) — для отображения графовой структуры программы. На первом этапе трансляции (front-end) строится абстрактное синтаксическое дерево (АСД). Затем в зависимости от сценария работы компилятора строятся так называемые графовые тени (ГТ), ориентированные на явные описания свойств транслируемых программ, например: граф зависимостей по данным, граф программных зависимостей, SSA-форма и т.д. Абстрактный синтаксический граф является объединением АСД и графовых теней, при котором к АСД добавляются новые графовые объекты (вершины и ребра) с особыми метками, сигнализирующими о принадлежности данного объекта некоторой графовой тени. При программной реализации ГТ должен быть предоставлен метод ее восстановления из АСГ.

Основной целью при проектировании ВП являлась его расширяемость. Поэтому ВП организовано в виде иерархии классов и библиотеки работы с ними. При разработке новых функциональных блоков системы пользователь может либо по-своему интерпретировать зарезервированное поле, присутствующее в любом из классов, либо наследовать от уже существующих классов новые типы объектов.

Данное ВП спроектировано в объектно-ориентированном стиле, библиотека работы с ними реализована на языке C++.

4. ЗАКЛЮЧЕНИЕ

На данном этапе авторами проекта реализовано ВП системы, а также переработана часть алгоритмов потокового анализа и построения теоретико-графовых представлений программ. Реализованы некоторые инструментальные компоненты: текстовый и графовый редакторы. Дальнейший этап

работ будет связан с формализацией и реализацией языка сценариев системы, с учетом тех целей, которые были ему заданы, а именно — поддержка *динамичности* по отношению к компонентам системы и элементам внутреннего представления. Выход предварительной версии системы предполагается к началу второго полугодия 2002 года. Она должна быть доступна через сайт лаборатории.

Особые слова благодарности хочется выразить научному руководителю д.ф-м.н. Евстигнееву В.А.

СПИСОК ЛИТЕРАТУРЫ

1. **Kasyanov V.N., Evstigneev V.A.** The PROGRESS program manipulation system // Parallel Computer Technologies/ Proc. Intern. Conf., Obninsk, 1993. — Vol. 3. — P. 651–656.
2. **Kasyanov V.N., Evstigneev V.A., Gorodniaia L.** The system PROGRESS is a tool for parallelizing compiler prototyping // Proc. of Eighth SIAM Conf. on Parallel Processing for Scientific Computing (PPSC-97), Minneapolis, 1997. — P. 301–306.
3. **Wolfe M.** High performance compilers for parallel computing. — Addison-Wesley, 1996. — P. 570.
4. **Aho A.V., Sethi R., Ullman J.D.** Compilers: Principles, Techniques and Tools. – Addison-Wesley, 1986.
5. **Wirth N.** Compiler construction. — Addison-Wesley, 1996. — P. 176.
6. **Касьянов В.Н. , Поттосин И.В.** Методы построения трансляторов. — Новосибирск: Наука, 1986. — 344 с.
7. **Логачева С.А.** Анализ зависимостей по данным на базе алгоритма Шостака // Поддержка супервычислений и интернет-ориентированные технологии. — Новосибирск, 2001. — С. 31–43.
8. **Шелехов В.И.** Внутреннее представление программ в системе Сократ. — Новосибирск, 1993. — 44 с. — (Препр./ РАН. Сиб. отд-ние. ИСИ; № 15.)
9. **Евстигнеев В.А.** О некоторых формах промежуточного представления программ //Конструирование и оптимизация программ. — Новосибирск, 1993. — С. 60–68.
10. **Маркин В.А.** Промежуточное представление программ в распараллеливающих компиляторах // Проблемы систем информатики и программирования. — Новосибирск, 1999. — С. 163–182.
11. **Евстигнеев В.А., Касьянов В.Н.** Сводимые графы и граф-модели в программировании. — Новосибирск: ИДМИ, 1999. — 288 с.
12. **Евстигнеев В.А., Касьянов В.Н.** Теория графов: алгоритмы обработки бесконтурных графов. — Новосибирск: Наука, 1998. — 385 с.
13. **Маркина С.А., Маркин В.А.** Проект системы для создания прототипа распараллеливающего компилятора. Расширяемое внутреннее представление системы

// Материалы конф. молодых ученых по математике, математическому моделированию и информатике, 4-6 декабря 2001 г. — Новосибирск, 2001. — С. 47–48.

14. **Евстигнеев В.А.** VLIW-машины: развитие архитектуры и принципов построения программного обеспечения // Системная информатика. Вып. 4: Методы теоретического и системного программирования. — Новосибирск: Наука, 1995. — С. 304–333.
15. **Евстигнеев В.А.** NUMA-архитектура: некоторые особенности компиляции и генерации кода // Поддержка супервычислений и интернет-ориентированные технологии. — Новосибирск, 2001. — С. 44–53.
16. **Евстигнеев В.А.** Анализ зависимостей: состояние проблемы // Системная информатика. Вып. 7: Проблемы теории и методологии создания параллельных и распределительных систем. — Новосибирск: Наука, 2000. — С. 112–173.

Л. С. Мельников, И. В. Петренко

ПУТЕВЫЕ ЯДРА И ДЛИНЫ ЦИКЛОВ В НЕОРИЕНТИРОВАННЫХ ГРАФАХ*

1. ВВЕДЕНИЕ

Пусть $G = (V, E)$ — конечный неориентированный граф. Число вершин в наиболее длинном пути графа G будем обозначать через $\tau(G)$. Через $g(G)$ и $c(G)$ будем обозначать длину кратчайшего (т. е. *обхват*) и длиннейшего циклов в G соответственно. Через C_n и P_n будем обозначать цикл длины n и путь с n вершинами соответственно. В графе G вершину $v \in V$ будем называть *P_n–терминальной вершиной* , если она является конечной вершиной пути P_n, но не является конечной для пути P_{n+1} в G.

Открытой окрестностью вершины $v \in V(G)$ называется множество вершин $N(v) = \{u \in V(G) | (u, v) \in E(G)\}$. *Открытой окрестностью подмножества* $A \subset V(G)$ называется множество

$$N(A) = \bigcup_{a \in A} N(a),$$

а *замкнутой окрестностью* множества A называется множество $N[A] = N(A) \cup A$. Подграф графа G, порожденный множеством S, будем обозначать через $G[S]$.

Введем следующие определения.

Определение 1. *Для произвольного графа G подмножество $K \subseteq V(G)$ называется P_n–ядром графа G, если $\tau(G[K]) \leq n - 1$ и каждая вершина $v \in V(G - K)$ смежна с P_{n-1}–терминальной вершиной из $G[K]$.*

Отметим, что максимальное независимое множество вершин образует P_2–ядро, а вершины максимального подграфа, не содержащего P_3, образуют P_3–ядро.

*Исследование первого автора поддержано Российским фондом фундаментальных исследований (гранты № 02-01-00039 и 00-07-90296), INTAS 97-1001 и Министерством образования РФ. Исследование второго автора поддержано Российским фондом фундаментальных исследований (грант № 99-01-00581).

Определение 2. *Для произвольного графа G подмножество $S \subseteq V(G)$ называется P_n–полуядром графа G, если $\tau(G[S]) \leq n-1$ и каждая вершина $v \in N(S) - S$ смежна с P_{n-1}–терминальной вершиной графа $G[S]$.*

Наиболее активно путевые ядра используются в связи с задачей о так называемом путевом разбиении произвольного графа. Пусть a, b — натуральные числа такие, что $a+b = \tau(G)$. Разбиение $\{A, B\}$ множества $V(G)$ называется (a, b)*–разбиением*, если $\tau(G[A]) \leq a$ и $\tau(G[B]) \leq b$. Если в графе G имеется (a, b)–разбиение для любой пары чисел (a, b) такой, что $a + b = \tau(G)$, то граф G называется τ*–разбиваемым.*

Известно, что наличие P_n–ядра в графе G означает, что G является $(\tau(G) - n + 1, n - 1)$–разбиваемым.

В [1] была высказана гипотеза о том, что произвольный граф G является τ–разбиваемым. Эта гипотеза также тесно связана с открытой проблемой Михока [11] о ядрах и нашла отражение в диссертациях [8] и [13]. Краткое описание рассматриваемых задач и их связь с отмеченной выше гипотезой имеется в [5]. Версия этой гипотезы для ориентированных графов имеется в [9].

Путевые разбиения в свою очередь тесно связаны с так называемым k*-хроматическим числом* $\chi_k(G)$ графа G, которое равно наименьшему количеству множеств $\{V_1, V_2, \ldots, V_n\}$, на которые разбивается $V(G)$ таким образом, что $\tau(G[V_i]) \leq k$ для каждого i. Оно было введено в [6], где также была получена верхняя оценка для k-хроматического числа произвольного графа: $\chi_k(G) \leq \lfloor(\tau(G) - 1 - k)/2\rfloor + 2$. Если гипотеза о τ–разбиваемости произвольного графа верна, то эта оценка может быть улучшена до $\chi_k(G) \leq \lceil\tau(G)/k\rceil$.

В [7] доказано, что если любой граф G из некоторого наследственного класса $\mathcal{H}$ имеет P_n–полуядро, то любой граф из $\mathcal{H}$ имеет P_n–ядро. Таким образом, утверждения о существовании в произвольном графе P_n–ядра и P_n–полуядра равносильны. Там же получены результаты о том, что если C является $(n-1)$–циклом в графе G, то C является P_n–полуядром графа G, а также, что если G — такой граф, что $g(G) \geq n-2$, то в G существует P_n–ядро.

Практический интерес представляет собой проблема о существовании P_n–ядра в произвольном графе. Однако решения этой проблемы в общем виде пока не получены, а доказательства для малых значений n имеют переборный характер и очень трудоемки. В [14] доказано, что в любом графе содержится P_8–ядро, причем методика доказательства

указывает на тесную взаимосвязь путевых ядер и длин циклов в неориентированных графах.

2. ПОСТРОЕНИЕ P_{N+2}–ЯДРА В ГРАФЕ С МАКСИМАЛЬНЫМ ЦИКЛОМ ДЛИНЫ N

Пусть G – конечный неориентированный граф и в нем имеется цикл C_n (т.е. цикл на n вершинах), причем $n = c(G)$. Исходя из утверждений, приведенных выше, для того, чтобы доказать, что в графе G имеется P_{n+2}–ядро, нам достаточно показать, что в графе G имеется P_{n+2}–полуядро.

Рассмотрим алгоритм $\mathcal{A}$, при помощи которого осуществляется построение P_{n+2}–ядра в данном графе G. Положим $S = C_n$.

Алгоритм $\mathcal{A}$. На начальном этапе для подмножеств A и B множества вершин V графа G положим $A = \emptyset$, $B = V(G) \setminus S$.

Шаг 1. Все вершины из B смежные с P_{n+1}–терминальными вершинами из множества S переместим в множество A. Если $N(S) \cap B = \emptyset$, то алгоритм завершает работу. В противном случае выполняется шаг 2.

Шаг 2. Если в множестве B имеется вершина b смежная с вершинами x и y из множества S, то вершина b перемещается в S, после чего выполняется шаг 1. В противном случае выполняется шаг 3.

Шаг 3. Если вершина x из S является P_n–терминальной и имеется вершина b из множества B, смежная с вершиной x, то вершина b перемещается в множество S, после чего выполняется шаг 1.

Все вершины множества S в начале работы алгоритма являются P_n–терминальными. Когда алгоритм завершит работу (т.е. когда множество $N(S) \cap B$ окажется пустым), множество S будет обладать свойствами P_{n+2}–ядра графа G. Каждая вершина из множества A будет смежна с P_{n+1}–терминальной вершиной из множества S. Таким образом, чтобы доказать, что S является P_{n+2}–ядром графа G, достаточно убедиться в том, что $\tau(S) \leq n + 1$.

Легко видеть, что путь длины большей, чем $n + 1$, в множестве S может возникнуть только при перемещении вершин из множества B в множество S. При выполнении шага 3 путь длины $n + 2$ и более возникнуть не может, так как шаг 3 выполняется тогда и только тогда, когда в множестве B не имеется вершин, смежных более, чем с одной вершиной из S. Следовательно, достаточно убедиться в том, что путь длины $n+2$ и более не возникнет при выполнении шага 2 алгоритма $\mathcal{A}$.

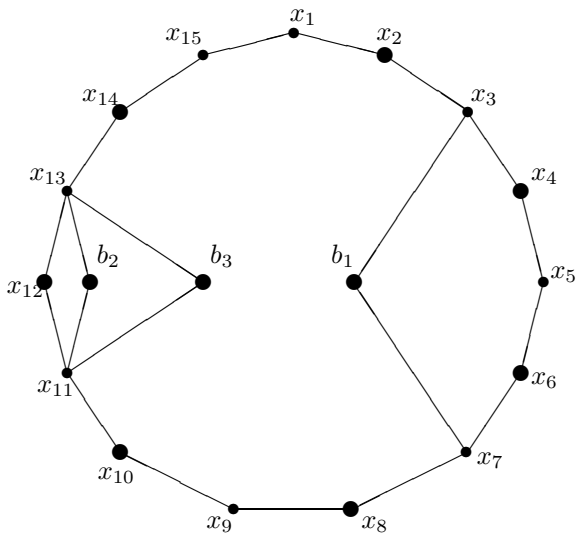


Рис. 1. $c(G) = n = 15$. Согласно лемме 2.1 вершины x_2, x_4, x_6, x_8, b_1, а также $x_{10}, x_{12}, x_{14}, b_2, b_3$ являются P_n–терминальными. Вершины b_2 и b_3 *дублируют друг друга*

Лемма 2.1. *Пусть G — граф, удовлетворяющий вышеперечисленным условиям, S – подграф графа G, в котором содержится C_n. Предположим, при выполнении шага 2 алгоритма $\mathcal{A}$ в множество S перемещена вершина b, смежная с вершинами x_i и x_j из множества S. Тогда вершины $x_{i-1}, x_{i+1}, x_{j-1}, x_{j+1}$, а также вершина b являются P_{n+1}–терминальными в подграфе $S \cup \{b\}$.*

Доказательство. Не ограничивая общности, можем предположить, что $1 < i < j < n$. Отметим также, что равенство $i + 1 = j$ не может выполняться ввиду предположения о том, что $n = c(G)$, т.е. n — длина наибольшего цикла в графе G. Вершины x_i и x_j являются P_n–терминальными в S. После перемещения вершины b в множество S получим, что в S возникли пути P_1, P_2, P_3, P_4 следующего вида:

$$P_1 = b, x_i, x_{i-1}, \ldots, x_{j+1}, x_j, x_{j-1}, \ldots, x_{i+1},$$
$$P_2 = b, x_i, x_{i+1}, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_{i-1},$$
$$P_3 = b, x_j, x_{j-1}, \ldots, x_{i+1}, x_i, x_{i-1}, \ldots, x_{j+1},$$

$$P_4 = b, x_j, x_{j+1}, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_{j-1},$$

длина каждого из этих путей равна $n+1$, и, следовательно, вершины $x_{i-1}, x_{i+1}, x_{j-1}, x_{j+1}$, а также вершина b являются P_{n+1}–терминальными в подграфе $S \cup \{b\}$. Лемма доказана.

Отметим, что, вообще говоря, возможно равенство $x_{i+1} = x_{j-1}$, и в этом случае лемма также справедлива (см. рис. 2, $x_i = x_{11}, x_j = x_{13}$).

Доказанная лемма позволяет сделать вывод о том, что после перемещения в множество S вершины b из множества B на шаге 2 при последующем *обязательном* выполнении шага 1 все вершины из B смежные с P_{n+1} – терминальными вершинами из S, будут перемещены в множество A.

В случае, если при двукратном выполнении шага 2 алгоритма $\mathcal{A}$ в множество S будут перемещены вершины b_1 и b_2, каждая из которых смежна с вершинами x_i и x_j из множества S, вершины $x_{i-1}, x_{i+1}, x_{j-1}$, x_{j+1}, b_1, b_2 являются P_{n+1}–терминальными в подграфе $S \cup \{b_1, b_2\}$. Такие вершины называются *дублирующими* друг друга, или *дубликатами.* На рис. 2 такими вершинами являются b_2 и b_3. Легко видеть, что перемещение в множество S в ходе работы алгоритма вершин, дублирующих друг друга, не оказывает влияния на $\tau(S)$. Действительно, поскольку путь максимальной длины в $S \cup \{b_1, b_2\}$ не может проходить через одну вершину более одного раза, то длина пути проходящего через вершины b_1 и b_2, дублирующие друг друга, не может превышать $n+1$. Это наблюдение может быть сформулировано в виде следующего утверждения.

Утверждение 2.2. *Пусть G — граф, удовлетворяющий вышеперечисленным условиям, S – подграф графа G, в котором содержится C_n. Пусть при многократном выполнении шага 2 алгоритма $\mathcal{A}$ в множество S перемещаются вершины $b_1, \ldots, b_l$, смежные с вершинами x_i и x_j из множества S и дублирующие друг друга. Тогда при перемещении вершин $b_2, \ldots, b_l$ в множество $S \cup \{b_1\}$ длина наибольшего пути в S не изменится.*

Это утверждение позволяет нам в дальнейшем не рассматривать вершины, дублирующие друг друга, считая их тождественными, что в свою очередь несколько упрощает доказательство следующей теоремы.

Теорема 2.3. *Пусть G — граф, удовлетворяющий вышеперечисленным условиям, $n = c(G)$, тогда в графе G имеется P_{n+2}–ядро.*

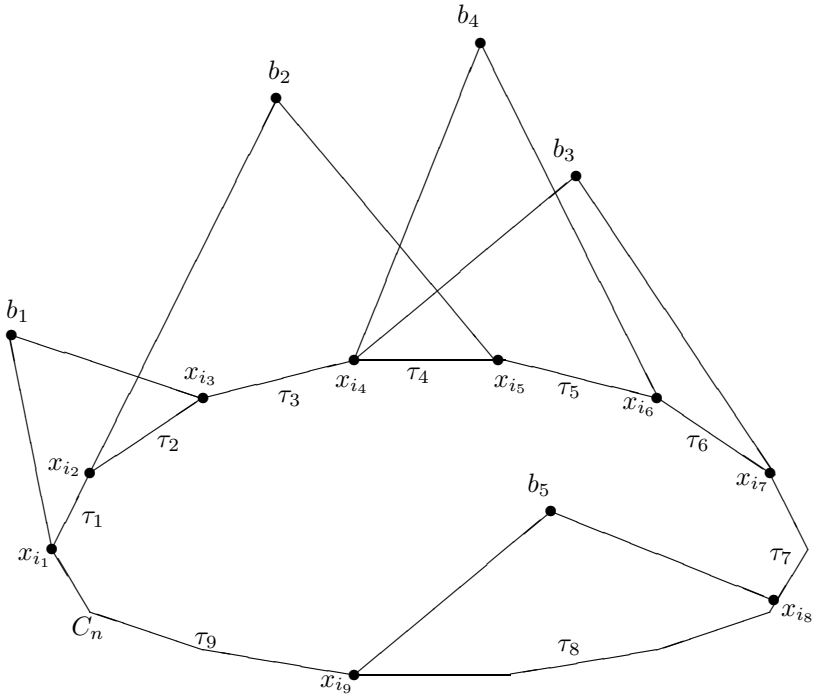


Рис. 2. Иллюстрация к доказательству теоремы 2.3, случай $k = 5, t = 9$

Доказательство. Как уже отмечалось выше, для доказательства теоремы достаточно показать, что при многократном выполнении шага 2 алгоритма $\mathcal{A}$ в S не образуется путь длины $n + 2$. Поскольку вершины, перемещенные в S при выполнении шага 3, путь такой длины порождать не могут, предполагаем, что при выполнении шага 3 ни одна вершина из B не была перемещена в множество S.

Пусть S — подграф графа G, в котором содержится C_n. Предположим, в процессе работы алгоритма $\mathcal{A}$ в множество S были перемещены вершины $b_1, b_2, \ldots, b_k$. Предположим также, что никакие две из этих вершин не являются дублирующими друг друга. Множество этих вершин обозначим через W.

В множестве $N(W) \cap S$ содержится t вершин цикла C_n. Обозначим их $x_{i_1}, x_{i_2}, \ldots, x_{i_t}$. Легко видеть, что при удалении этих вершин цикл C_n

распадается на t цепей $p_1, p_2, \ldots, p_t$, где $t \leq 2k$, причем длина каждой из этих цепей $\tau_1, \tau_2, \ldots, \tau_t$ соответственно. В силу леммы 2.1 $\tau_i \geq 1$, где $i = 1, \ldots, t$.

Пусть P — путь, который проходит через каждую из вершин множества W и имеет наибольшую длину в S. Поскольку P — простой путь и проходит через каждую из содержащихся в нем вершин в точности один раз, его длина может быть легко подсчитана:

$$\tau(P) = \sum_{i \in I} \tau_i + t + k,$$

где $I \subseteq \{1, 2, \ldots, t\}$, причем $i \in I$ тогда и только тогда, когда путь P проходит через вершины цепи p_i. Нетрудно видеть, что $|I| \leq t - k + 1$.

Через J обозначим множество $\{1, 2, \ldots, t\} \setminus I$, т. е. множество номеров цепей $p_1, p_2, \ldots, p_t$, через вершины которых путь P не проходит. Отметим, что $|J| \geq k - 1$.

Поскольку имеет место равенство

$$\sum_{i=1}^{t} \tau_i + t = n,$$

которое можно также записать в виде

$$\sum_{i \in I} \tau_i + \sum_{j \in J} \tau_j + t = n,$$

имеем

$$\sum_{i \in I} \tau_i = n - \sum_{j \in J} \tau_j - t.$$

Тогда длина пути P может быть подсчитана следующим образом:

$$\tau(P) = \sum_{i \in I} \tau_i + k + t = n - \sum_{j \in J} \tau_j + k.$$

Поскольку $\tau_i \geq 1$, где $i = 1, 2, \ldots, t$, а также $|J| \geq k - 1$ получим:

$$\tau(P) = n - \sum_{j \in J} \tau_j + k \leq n - (k - 1) + k = n + 1,$$

что и требовалось доказать.

3. ЗАКЛЮЧЕНИЕ

Авторы считают необходимым отметить, что примененная в настоящей статье техника может быть использована также для доказательства того, что *если в неориентированном графе G имеется цикл C_n и $n = c(G) \leq 7$, то в G имеется P_{n+3}–ядро* . Доказательство это, однако, чрезмерно громоздкое и трудоемкое из-за своего переборного характера. В связи с этим авторы позволили себе ограничиться здесь лишь некоторыми соображениями, не приводя доказательства полностью.

Техника его в целом аналогична применяемой в настоящей статье, необходимо лишь слегка модифицировать алгоритм $\mathcal{A}$, который примет при этом следующий вид.

На начальном этапе для подмножеств A и B множества вершин V графа G положим $A = \emptyset$, $B = V(G) \setminus V(S)$.

Шаг 1. Все вершины из B, смежные с P_{n+2}–терминальными вершинами из множества S, переместим в множество A. Если $N(S) \cap B = \emptyset$, то алгоритм завершает работу. В противном случае выполняется шаг 2.

Шаг 2. Если в множестве B имеется вершина b, смежная с вершинами x и y из множества S, то вершина b перемещается в S, после чего выполняется шаг 1. В противном случае выполняется шаг 3.

Шаг 3. Если вершина x из S является P_{n+1}–терминальной и имеется вершина b из множества B, смежная с вершиной x, то вершина b перемещается в множество S, после чего выполняется шаг 1. Иначе выполняется шаг 4.

Шаг 4. Если вершина x из S является P_n–терминальной и имеется вершина b из множества B, смежная с вершиной x, то вершина b перемещается в множество S, после чего выполняется шаг 1. В противном случае выполняется шаг 2.

Исследуя работу этого алгоритма на всех возможных подграфах S графа G, содержащих C_n с $\tau(S) \leq n + 2$ (а таких подграфов в случае $n = 7$ окажется 16), получим доказательство.

Ограничение на значение $n = c(G)$ в известной степени условно. Доказать существование P_n–ядра для больших значений n, используя этот метод, также представляется возможным. Однако число рассматриваемых подграфов с ростом n возрастает очень стремительно, в связи с чем использование данной методики неэффективно. К тому же выводу нас приводит и то, что в случае, когда $n = 15$, в процессе работы алгоритма в множество S могут быть добавлены вершины b_1, b_2, b_3 из

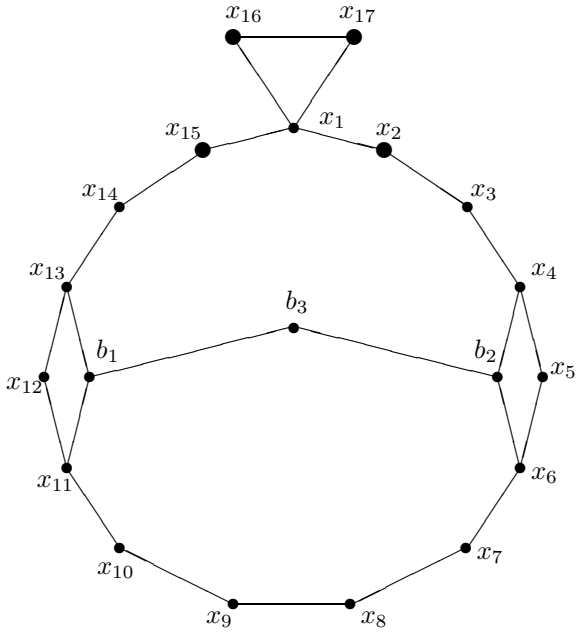


Рис. 3. В S имеется путь длины 18

множества B, при этом в S образуется путь длины 18, в то время как для существования P_{n+3}–ядра (в данном случае — P_{18}–ядра) необходимо выполнение неравенства $\tau(S) \leq 17$. Эта ситуация изображена на рис. 3.

Изложенные в предыдущем абзаце рассуждения не позволяют построить пример неориентированного графа, не имеющего путевого ядра. Более того, предположение о существовании такого графа представляется неправдоподобным.

Авторы полагают, что возможности использованного метода построения путевого ядра в неориентированном графе G на этом исчерпываются, допуская однако, что некоторая дополнительная модификация алгоритма $\mathcal{A}$ за счет структурирования шага 2 позволит доказать утверждение о существовании P_{n+3}–ядра в неориентированном графе G с циклом максимальной длины n и для значений $n > 7$. Следует ожидать,

что трудоемкость доказательства при этом многократно возрастет за счет необходимости перебора множества дополнительных «подшагов» шага 2.

СПИСОК ЛИТЕРАТУРЫ

1. **Borowiecki M., Broere I., Frick M., Mihók P., Semanišin G.** A survey of hereditary properties of graphs // Discuss. Math. Graph Theory. — 1997. — Vol. 17, N 1. — P. 5–50.
2. **Broere I., Dorfling M., Dunbar J. E., Frick M.** A path(ological) partition problem // Discuss. Math. Graph Theory. — 1998. — Vol. 18, № 1. — P. 113–125.
3. **Broere I., Dorfling M.** The decomposibility of additive of hereditary properties of graphs // Discuss. Math. Graph Theory. — 2000. — Vol. 20, N 2. — P. 281–291.
4. **Broere I, Frick M., Semanišin G.** Maximal graphs with respect to hereditary properties // Discuss. Math. Graph Theory. — 1997. — Vol. 17, N 1. — P. 51–66.
5. **Broere I., Hajnal P., Mihók P.**. Partition problems and kernels of graphs // Discuss. Math. Graph Theory. — 1997. — Vol.17, N 2. — P. 51-56.
6. **Chartrand G., Geller D. P., Hedetniemi S. T.**, A generalization of the chromatic number // Proc. Camb. Phil. Soc. — 1968. — Vol. 64, N 2. — P. 265–271.
7. **Dunbar J. E., Frick M.** Path kernels and partitions // J. Combin. Math. Combin. Comput. — 1999. — Vol. 31. — P. 137–149.
8. **Hajnal P.** Graph partitions (на венгерском): Ph. D. Thesis. — Szeged Attila József University, 1984.
9. **Laborde J. M., Payan C., Xuong N. H.** Independent sets and longest directed paths in digraphs // Graphs and other combinatorial topics (Prague, 1982). — Leipzig: Teubner, 1983. — P. 173–177.
10. **Lovász L.** On decomposition of graphs // Studia Sci. Math. Hungar. — 1966. — Vol. 1, N 1–2. — P. 237–238.
11. **Mihók P.** Problem 4. Graphs, hypergraphs and matroids. — Zielona Góra: Higher College Engrg., 1985. — 86 p.
12. **Stiebitz M.** Decomposing graphs under degree constraints // J. Graph Theory. — 1996. — Vol. 23, N 3. — P. 321–324.
13. **Vronka J.** Vertex sets of graphs with prescribed properties (на словацком): Ph. D. Thesis. — Košice, Šafárik University, 1986.
14. **Мельников Л.С., Петренко И.В.** Путевые ядра и разбиения в неориентированных графах //Дискретный анализ и исследование операций. Сер. 1. — 2002. — Т.9, № 2. — С. 21–35.

Д. Ю. Мехонтцев, И. В. Лобив, К. С. Селезнев

СЛЕЖЕНИЕ И ОПРЕДЕЛЕНИЕ СКОРОСТИ ДВИЖУЩИХСЯ НА ПЛОСКОСТИ ОБЪЕКТОВ В РЕАЛЬНОМ ВРЕМЕНИ*

1. ВВЕДЕНИЕ

В последнее время в связи с ростом мощности компьютеров появилась возможность создания относительно дешевых программно–аппаратных систем, реализующих обработку большого количества информации, поступающей в реальном времени. Примером может служить система отслеживания движения объектов, на вход которой поступают изображения, полученные с видеокамеры, а на выходе имеются данные об объектах, попавших в поле зрения, а также об их динамических характеристиках, например, скорости и ускорении.

Нашей группой успешно реализована система такого рода, в которой в качестве аппаратной части выступает обычный персональный компьютер класса Pentium-2, а в качестве программной части используется логический модуль нашей собственной разработки, устройству которого и посвящена данная статья.

К особенностям системы можно отнести следующие пункты:

1) камера должна обозревать от кадра к кадру одну и ту же область пространства;
2) допускаются цветовые шумы и небольшое дрожание камеры;
3) отслеживаемые объекты могут плавно менять форму и размер (например, приближающийся автомобиль);
4) не допускается пересечение движущихся объектов одинакового цвета на продолжительное время.

2. ПОСТАНОВКА ЗАДАЧИ

Рассмотрим декартову систему координат (x, y, z). Возьмем некоторую точку $S(0,0,h_0)$. Под углом α через точку S проведена плоскость (плос-

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

кость экрана) параллельно оси y. На этой плоскости зададим экран — прямоугольник длиной L (параллельной оси y), высотой H и с пересечением диагоналей в точке S. Глаз наблюдателя E находится на расстоянии d от точки S плоскости экрана. По плоскости (x, y) вдоль вектора V движутся некоторые трехмерные объекты.

Наша задача — определить их мгновенную и среднюю скорость при прохождении в поле видимости. Расчеты должны происходить в реальном времени.

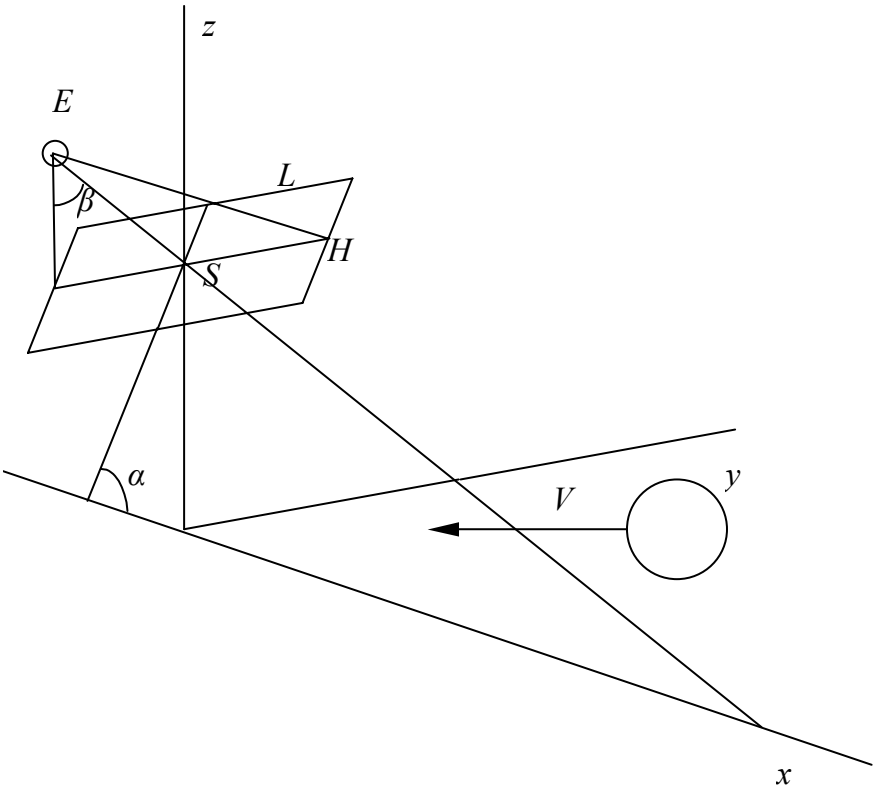


Рис. 1. Постановка задачи

3. БЛОК - СХЕМА

4. ОПИСАНИЕ РАБОТЫ АЛГОРИТМОВ

На вход логическому модулю подается последовательность изображений (bitmap sequence). Требуется некоторое время (10-20 кадров) для расчета дороги (автонастройки). Затем выделяются движущиеся объекты на изображении, и формируются цепочки. По мере поступления изображений (bitmaps) рассчитывается их скорость. В момент достижения объектом конца области видимости логический модуль добавляет его к списку найденных объектов.

Этап 0. Начало работы, инициализация алгоритмов.

Этап 1. Расчет «дороги».

Алгоритм расчета дороги состоит в следующем. Берем из потока несколько, желательно как можно более различных, кадров (для того, чтобы дорога определилась без помех). Далее для каждого пиксела проводим на всех взятых кадрах попарные сравнения и выявляем такие пикселы (pixels), которые не двигаются — это пикселы, принадлежащие «дороге», а остальные принадлежат машинам. Если на выходе получилась «дорога» без «дырок», то значит, она нашлась вполне удовлетворительно и вероятность ошибки снижается. Заметим, что эта процедура довольно трудоемкая, но она выполняется редко: один раз при настройке (например, один раз в час либо при резком изменении погодных условий).

Мы испробовали множество алгоритмов. В некоторых ситуациях подходит один (к примеру, когда требуется провести процедуру как можно быстрее), в других ситуациях — другой. Мы опишем только два: самый простой и тот, который используется в настоящее время.

Самый простой способ расчета «дороги».

Возьмем последовательность фреймов (frame) $F_0,...,F_{N-1}$. Определим фрейм дороги R, т.е. фрейм, на котором нет ни одной движущейся области — «машины». (Из этого, кстати, следует, что стоящие на обочине машины для нас тоже являются частью дороги, из-за их неподвижности в момент расчета).

Итак, введем вспомогательное множество D, в котором мы будем хранить пары вида (x, y), если мы уже приняли решение о цвете пикселя (x, y) фрейма «дорога». Соответственно, цвет пикселов, пары которых не включены в множество, находится под вопросом.

$D = 0,$ т.е. множество пикселов дороги изначально пусто;

$R_0 = F_0,$ нулевое приближение — это 0-й фрейм;

$R_1 = R_0 - F_1,$ первое приближение — из нулевого «вычесть» первый фрейм;

.....

$R_{N-1} = R_{N-2} - F_{N-1};$

где разность фреймов, $A - B$, определяется следующим образом:
$\langle i, j \rangle \in D \Rightarrow a_{ij} - b_{ij} = a_{ij},$

$$\langle i,j\rangle \notin D \Rightarrow a_{ij} - b_{ij} = \begin{cases} a_{ij}, \text{если } a_{ij} == b_{ij}. \text{В этом случае } D = D \cup (i,j); \\ \text{иначе неопределено.} \end{cases}$$

Сравнение двух цветов ($a_{ij} == b_{ij}$) мы ведем в некоторой цветовой норме, например,

$$c^1 == c^2 = \begin{cases} 1, \text{ если } m < C; \\ 0, \text{ иначе,} \end{cases}$$

где $m = \left|c_R^1 - c_R^2\right| + \left|c_G^1 - c_G^2\right| + \left|c_B^1 - c_B^2\right|$, C — константа.

В итоге получим

$$R = R_N = \begin{cases} r_{ij}^{N-1}, \text{если } r_{ij}^{N-1} \text{ определен;} \\ \mathrm{f}_{\mathrm{ij}}^0, \text{иначе.} \end{cases}$$

Легко увидеть основной недостаток данного алгоритма — фрейм «дороги» будет очень похож на 0-й фрейм, чуть меньше — на 1-й фрейм, ..., меньше всего — на $N-1$ фрейм, т.е. если 0-й фрейм последовательности мы взяли неудачно, то и фрейм «дороги» будет очень не похож на реальную дорогу, и мы очень часто будем ошибаться в выделении «машин».

Текущий способ расчета «дороги».

Рассмотрим последовательность цветов точки с координатами (x,y) на всех N фреймах, которые даны для вычисления «дороги», т.е. $c_0,\ldots,c_{N-1}$ — это соответствующие цвета. Рассмотрим подпоследовательность этой последовательности $c_j,\ldots,c_k$, где $0 \le j < N$, $j+1 < k < N$, такую что

$$c_j \neq c_{j+1},$$
$$c_j \neq c_{j+2},$$
$$\ldots,$$
$$c_j \neq c_{k-1},$$
$$c_j = c_k.$$

Результирующий цвет дороги в точке с координатами (x,y) будет цвет c_j, если такая подпоследовательность найдена, и c_0 — иначе.

Другими словами, первый цвет, который удовлетворил:

– первый раз нам встретился,
– исчез в одном или нескольких фреймах,
– далее он опять встретился.

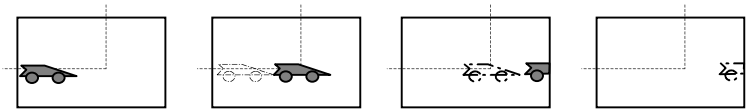


Рис. 2. Движение машины относительно точки (x, y).
Пунктиром показано положение машины на предыдущем фрейме

Этот способ расчета дороги очень хорошо согласуется с реальной жизнью: сначала на некоторое время дорога пуста, потом по ней быстро проезжает машина, и далее опять — начальное состояние. Конечно, можно возразить, а что будет, если друг за другом едут две, к примеру, синих машины. В этом случае можно попробовать посчитать, сколько из N цветов одинаковых, и принять этот цвет за дорогу. В наших экспериментах текущий алгоритм вполне хорошо себя зарекомендовал, если брать фреймы для дороги через достаточно большой промежуток времени друг от друга (мы брали через каждые 10 фреймов).

Этап 2. Начало цикла по кадрам. На вход с камеры начинают поступать изображения (bitmaps).

Этап 2.1 Предобработка кадра.

Этап локализации областей, где потенциально находятся движущиеся объекты

Берем текущий кадр и начинаем каждый пиксел сравнивать с «дорогой» и предыдущим кадром. Таким образом мы выделим области, где могут находиться машины и шумы. Кадр с помеченными областями назовем предобработанным. Опишем подробнее два алгоритма этого этапа локализации.

Простейший алгоритм локализации областей

Пусть F_i — текущий фрейм, а R — фрейм дороги. Рассмотрим цвет точки p с координатами (x, y) на этих фреймах, т.е. c_i, c_R соответственно.

Наша задача — понять, является ли точка p на фрейме F_i частью некоторого движущегося объекта или нет.

Пусть Ω_i — это объединение всех таких областей на фрейме F_i, тогда положим $p \in \Omega_i, \Leftrightarrow c_i \neq c_R$.

Улучшенный алгоритм локализации областей

Рассмотрим $\Omega'_{i-1} \subseteq \Omega_{i-1}$ — объединение всех областей, где точно находятся «машины» на предыдущем фрейме F_{i-1}.

Если $\Omega'_{i-1} = 0$, то пользуемся предыдущим алгоритмом.

В противном случае,

- если $p \in \Omega'_{i-1}$, то $p \in \Omega_i, \Leftrightarrow c_i \neq c_R$,
- если $p \notin \Omega'_{i-1}$, то $p \in \Omega_i, \Leftrightarrow c_i \neq c_{i-1}$.

Видно, что в данном алгоритме ответственность за решение лежит не только на фрейме дороги, но и на предыдущем фрейме. Это позволило всем расчетам стать более устойчивыми к таким явлениям, как дрожание камеры и др. (т.к. у нас все тесты были сняты вручную), и сгладило другие вредные эффекты (например, неудачное нахождение фрейма «дороги»).

Этап выделения контуров в областях, где потенциально находятся движущиеся объекты

Пусть K_j — множество пикселей j-го замкнутого контура. $Int(K_j)$ — внутренность K_j, т.е. все пиксели которые лежат внутри K_j, но не принадлежат ему. Пусть $K = \bigcup_j K_j$ — объединение контуров на текущем фрейме F_i, $Int(K) = \bigcup_j Int(K_j)$. Мы сканируем F_i в поисках пикселя $p \middle| p \notin K, p \notin Int(K)$, например, двигаем однопиксельный сканер слева направо, а сканируем сверху вниз. Пусть мы нашли такой пиксель, тогда он принадлежит к еще не обойденному контуру. Обойдем его, как показано на рис. 3.

Заметим, что этот алгоритм можно легко модифицировать таким образом, чтобы не обходить области толщиной в один или два пикселя (например, линию длиной в 5 пикселей на рис. 3). Такая модификация понадобилась нам для того, чтобы уменьшить влияние шумов, так как на первых тестах контуры были похожи на осьминогов с массивными телами и очень раз-

ветвленными щупальцами, по длине иногда даже в несколько раз превосходившие длину тела.

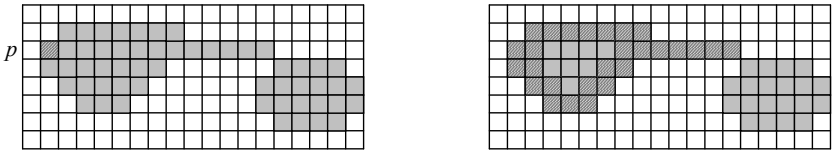


Рис. 3. Обхождение контура. p — «начало» контура.
Штриховкой показаны точки, которые принадлежат контуру

Таким образом мы выделим все контуры. Контуры у которых число точек мало или у них «неправильная» геометрия, являются шумами и отбрасываются. Оставшиеся контуры K_j являются некоторыми «машинами». Для них вычисляется ряд параметров (т.е. центр масс, угловые размеры) для того, чтобы подклеивать их в уже существующие цепочки «машин» D_m или создавать новую цепочку с этим головным элементом.

Итак, мы получили на этом этапе неупорядоченный массив K_j и полностью перешли от пространства пикселей к пространству контуров.

Этап 2.2. Начало цикла по контурам.

По одному начинаем рассматривать каждый контур K_j и соответствующий набор параметров (центр масс, угловые размеры, а также уже рассчитанные некоторые новые, такие как направление движения, среднюю скорость и др.).

Этап 2.2.1. Привязка контура к структуре цепочек.

Рассмотрим контур K_j и цепочку машин $D_z[1..n]$. Проверим, является ли K_j продолжением цепочки $D_z[1..n]$. Для этого сравним параметры и направление движения (машина, например, не может мгновенно изменить направление движения больше, чем на некоторый угол).

Этап 2.2.2. Анализ цепочки.

Пусть параметры K_j с наивысшей точностью совпали с параметрами цепочки $D_z[1..n]$. Тогда мы присоединяем K_j к этой цепочке и пересчитываем некоторые усредненные параметры, связанные с этой цепочкой.

Проанализируем, достигла ли машина конца области видимости. Если да, то замыкаем цепочку и проводим ее анализ на то, является ли она шумом. Если это не шум, значит вычисляем время движения, длину траектории, среднюю скорость и выводим эти данные.

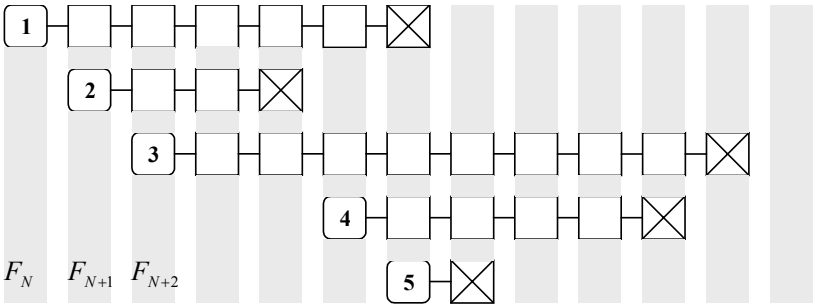


Рис. 4. Анализ цепочек (1..5), которые составлены из контуров (на рис. это прямоугольники), лежащих на фреймах $F_N, F_{N+1}, \ldots$

На рис. 4 показан пример такого анализа. Видно, что цепочки 2 и 4 возможно «склеить» (т.е. в конец цепочки 2 вставить цепочку 4), если у них совпадут средние параметры. Такого рода «склейки» применялись в нашем проекте, так как на некотором этапе мы пришли к тому, что цепочки из-за шумов и неточности вычислений рвались (т.е. начиналась новая цепочка вместо того, чтобы продолжить существующую). Из-за этого мы, бывало, «теряли» машину в центре области видимости. Цепочка 5 — это шум, и во время анализа мы ее вообще отбросим. По остальным рассчитаем среднюю скорость и интегральные параметры, если они еще не завершены.

5. РЕАЛИЗАЦИЯ

В качестве тестов были взяты несколько роликов движения машин по автостраде, отснятых цифровой камерой. Также были взяты несколько на-

боров входных параметров (для каждого ролика — свой набор, определяющий ранее введенные величины). Написан программный продукт "**HighWay**", реализованный на MS VC ++ 6.0 (MFC, C++, MS Direct Show). Он представлен в виде окна (на котором проигрывается avi-файл), списка изображений найденных объектов со скоростями, а также панели управления, с помощью которой можно задать входные параметры и выполнить команды: open video file, start, stop, pause, exit.

Сам продукт выполнен из двух независимых частей.

Логический модуль, в котором реализованы наши уникальные и общеизвестные алгоритмы, выполнен на C++.

Интерфейсная часть (оболочка) использует современные новейшие технологии обработки видеоинформации и написана на MFC и Windows SDK.

Запуск происходит в нескольких потоках: в первом работает распаковка видеофайла, во втором — центр управления, в третьем работает алгоритмическая часть.

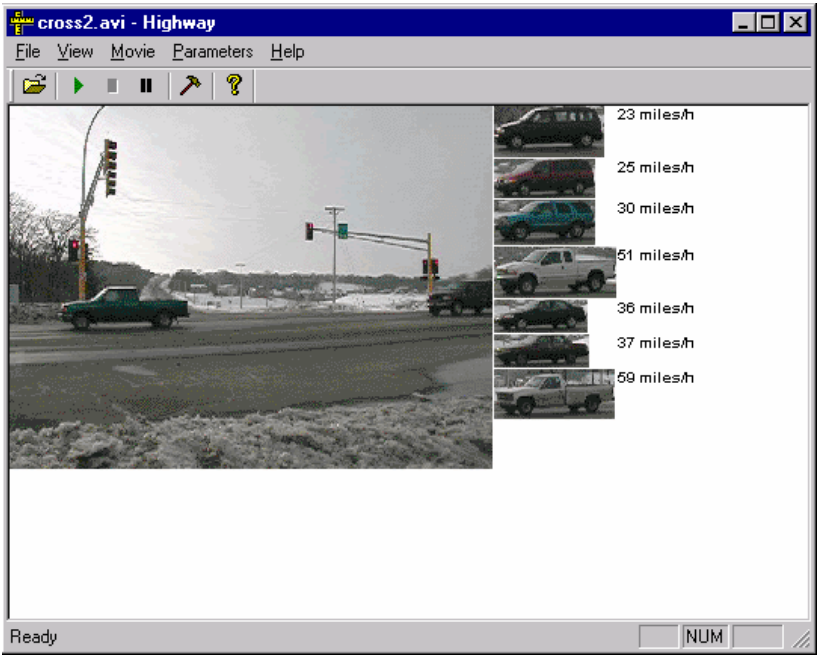


Рис. 5. Внешний вид продукта

Где можно взять для свободного просмотра продукт "HighWay"?
В разделе: http://gs.ieie.nsc.ru/projects/gso/html/gso/image/Light/

- Release 1.1.1 light.zip — сам продукт,
- cross2.zip — сжатый avi-файл теста,
- movie1.zip — сжатый avi-файл теста,
- movie2.zip — сжатый avi-файл теста,

а также
http://www.gadgetsoft.com/cgi-bin/default?page=home/divisions/image/highway

СПИСОК ЛИТЕРАТУРЫ

1. **Pratt, William K.** Digital image processing. 2nd ed.. — Wiley & Sons Ltd., 1991. — 438 p.
2. **Кнут Д.** Искусство программирования на ЭВМ. Т. 3: Сортировка и поиск. — М.: Мир, 1978.
3. The 4th All-Russian with invited foreign participants Conference «Pattern Recognition and Image Analysis: New Information Technologies» ROAI-98, Novosibirsk: IA&E SibRAS, 1998. — Vol. 1, 2.

В. И. Терехов, Н. Г. Треногин

ОПТИМАЛЬНОЕ РАЗМЕЩЕНИЕ ИНФОРМАЦИИ В ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ С УЧЕТОМ СТРУКТУРНОЙ НАДЕЖНОСТИ КОМПОНЕНТОВ*

Существует довольно большой класс вычислительных систем, построенных на базе серверов СУБД. При этом данные (таблицы) могут располагаться либо на одном сервере СУБД (централизованная система), либо на нескольких серверах, удаленных друг от друга (распределенная система).

В данной статье описан метод для оптимального распределения данных по серверам СУБД с учетом надежности системы.

Постановка задачи: рассматривается распределенная информационная система, узлы которой связаны между собой каналами глобальной сети. Каждый узел подключен к этой сети на определенной скорости. Узел состоит из локальной сети, к которой подключены пользователи, и сервера, хранящего данные пользователей.

Для моделирования описанной выше информационной системы предлагается использовать аналитическую модель, построенную на основе теории очередей.

Введем следующие обозначения:

N — количество узлов;

i, k — номер узла I = 1..N, k = 1..N (причем, пользователи i-го узла обращаются к узлу k);

n_i — число пользователей, подключенных к i-му узлу;

M — количество таблиц, которые предстоит оптимально распределить по серверам сети;

j — номер таблицы, j = 1...M;

V_j — объем j-й таблицы;

λ_j — интенсивность запросов пользователя к j-й таблице;

$$Y_{ijk} = \begin{cases} 1, & \text{åñëè пользователи i - го узла обращаются ê óçëó k çà j - é òàáëèöåé,} \\ 0 & \text{иначе;} \end{cases}$$

$$X_{ij} = \begin{cases} 1, & \text{если j - я таблица имеется на i - м узле,} \\ 0 & \text{иначе.} \end{cases}$$

* Работа выполнена при финансовой поддержке Министерства образования РФ.

Тогда $X_{ij} = Y_{ijk,I=k}$, т.е. если таблица j имеется на i-м узле, то пользователи обращаются к ней.

Кроме того, таблица должна быть хотя бы на одном из узлов, и пользователи каждого узла должны иметь доступ ко всем таблицам. В установившемся режиме пользователи i-го узла обращаются к одному конкретному k-му узлу за j-й таблицей.

Тогда для всех j и i имеем:

$$\sum_k Y_{ijk} = 1.$$

Интенсивность нагрузки, поступающей на сервер k-го узла, определяется как

$$\Lambda_{k\,вх} = \sum_j \sum_i Y_{ijk} \lambda_j n_i .$$

Часть этой нагрузки приходит от пользователей этого же узла k, а другая часть — по сети. Определим эту часть:

$$\Lambda'_{k\,вх} = \Lambda_{k\,вх} - n_k \sum_j X_{kj} \lambda_j .$$

Для каждого i-го узла интенсивность запросов к другим узлам сети $\Lambda'_{i\,исх}$ определяется как суммарная интенсивность от пользователей i-го узла $\Lambda_{iисх}$ за вычетом той доли, что замыкается в i-м узле:

$$\Lambda_{i\,исх} = \sum_j \sum_k Y_{ijk} \lambda_j n_i ;$$

$$\Lambda'_{i\,исх} = \Lambda_{i\,исх} - n_i \sum_j X_{ij} \lambda_j .$$

Нагрузка от i-го узла к k-му находится как

$$\Lambda_{ik} = n_i \sum_j Y_{ijk} \lambda_j .$$

Кроме того, могут быть заданы ограничения на объем хранимых данных в каждом узле

$$\sum_j X_{kj} V_j < V_{max} .$$

Время обслуживания на различных элементах СеМО зависит от интенсивности обслуживания в канале k-го узла и на сервере k-го узла (μ'_i и μ_i соответственно).

Нагрузка, входящая в k-й узел, складывается из ответов на запросы пользователей k-го узла к остальным узлам системы и запросов пользователей к серверу k-го узла. Считаем, что интенсивность ответов однозначно связана с интенсивностью запросов:

$$\Lambda'_{k\,вх\,\Sigma} = \Lambda'_{k\,вх} + f(\Lambda'_{k\,исх}).$$

Нагрузка, исходящая из узла k, складывается из запросов пользователей k-го узла к остальным узлам системы и ответов на запросы пользователям других узлов. Считаем, что интенсивность ответов однозначно связана с интенсивностью запросов:

$$\Lambda'_{k\,исх\,\Sigma} = \Lambda'_{k\,исх} + f(\Lambda'_{k\,вх}).$$

В простейшем случае:

$$\Lambda'_{k\,исх\,\Sigma} = \Lambda'_{k\,вх\,\Sigma} = \Lambda'_{k\,вх} + \Lambda'_{k\,исх}.$$

Для СМО M/M/1 время обслуживания на сервере и на канале в обоих направлениях равно

$$T_{k\,сер} = 1/(\mu_k - \Lambda_{k\,вх});$$
$$T_{k\,вх} = 1/(\mu'_k - \Lambda'_{k\,вх\,\Sigma});$$
$$T_{k\,исх} = 1/(\mu'_k - \Lambda'_{k\,исх\,\Sigma}).$$

Среднее время задержки для пользователей каждого из узлов может быть найдено как

$$T_k = ((\Lambda_{k\,исх} - \Lambda'_{k\,исх}) / \Lambda_{k\,исх})\ T_{k\,сер} + (\Lambda'_{k\,исх}/\Lambda_{k\,исх})T_{k\,исх} + \sum(\Lambda_{ik} / \Lambda_{k\,исх})(T_{i\,вх} + T_{i\,сер} + T_{i\,исх} + T_{k\,вх}).$$

Это время не должно быть больше заданного.

В приложении к расчетным системам операторов связи, данный поход освещен в [1].

Введем в данную модель дополнительные ограничения, учитывающие показатели надежности системы.

Пусть $K_{г\,сер\,i}$ — коэффициент готовности сервера i-го узла, $K_{г\,кан\,i}$ — коэффициент готовности канала i-го узла.

Рассматривая данную информационную систему с точки зрения надежности, можно выделить два случая.

1. В случае критического сбоя клиентское приложение имеет возможность переключаться и работать с другими копиями таблиц. Таким образом, в системе параллельно существуют несколько копий каждой таблицы. Для такого варианта правильнее рассматривать коэффициент готовности каждой j-й таблицы

$$K_{гj} = 1 - \prod_i (1 - K_{г\,сер\,i}\,K_{г\,кан\,i}X_{ij}).$$

Таким образом, чем больше копий таблицы будет размещено на различных узлах, тем выше коэффициент готовности. Данное соотношение можно рассматривать в виде дополнительного ограничения при поиске оптимального решения, если установить $K_{гj} > 0.98$.

2. Архитектура клиентского приложения неизменна, и переключение на другие копии невозможно. В данном случае надежность можно рассматривать с точки зрения отказа в обслуживании пользователям узла i.

$$K_{гi} = (1- (1- K_{г\ сер\ i})(1-\prod_{j} (1-X_{ij}))) \bullet \prod_{k,k\neq i} (1- (1- K_{г\ сер\ k}K_{г\ кан\ k})(1-\prod_{j} (1-Y_{ijk})))x(1-(1-K_{г\ кан\ i})(1-\prod_{j} X_{ij})).$$

В данной формуле коэффициент готовности системы для пользователей i-ого узла определятся как произведение коэффициента готовности сервера на узле i (при условии, что на этом узле используется хоть одна таблица), коэффициентов готовности серверов и каналов узлов k (если используются таблицы с узла k) и коэффициента готовности канала узла i (если используется хотя бы одна таблица вне i-ого узла). В данном случае надежность тем ниже, чем больше количество узлов, к которым необходимо обращаться.

Приведенное соотношение можно рассматривать в виде дополнительного ограничения при поиске оптимального решения, если установить $K_{гi} > 0.98$.

СПИСОК ЛИТЕРАТУРЫ

1. **Треногин Н. Г., Терехов В. И.** Методика размещения данных при проектировании биллинговых систем // Электросвязь. — 2002. — № 3. — С. 32–34.

СОДЕРЖАНИЕ

Предисловие редактора 5
Бабурин Д. Е. Иерархический подход для автоматического размещения ациклических графов 7
Волянская Т. А. Методы и технологии адаптивной гипермедиа 38
Глуханков М. П., Дортман П. А., Павлов А. А., Стасенко А. П. Транслирующие компоненты системы функционального программирования SFP 69
Дунаев А. А., Лобив И. В., Мехонцев Д. Ю., Мурзин Ф. А., Половинко О. Н., Семич Д. Ф., Чепель А. В., Ярков К. А. Алгоритмы быстрого поиска фрагментов фотографических изображений 88
Дылыков Ж. Л-Д., Мурзин Ф. А. Система TRIZ_Computing 110
Дылыков Ж. Л-Д., Пустыльников В. А. Автоматизация методов принятия решений на железнодорожном транспорте как гарантия обеспечения безопасности движения 120
Евстигнеев В. А. Конвейерная модель перевозок как модель пересылки протяженных сообщений 129
Евстигнеев В. А., Мирзуитова И. Л. Развитие NUMA-архитектуры: текущее состояние 139
Иванов М. А. Обзор MPEG-подобных методов кодирования видеоданных 155
Касьянов В. Н., Несговорова Г. П., Волянская Т. А. Виртуальный музей истории информатики в Сибири 169
Кряженков П. Б. Пользовательский интерфейс интегрированной среды функционального программирования SFP 182
Малинина Ю. В. ИС ТРАНСФОРМ: Автоматизация наполнения системы 201
Маркин В. А., Маркина С. А. Проект системы для быстрого прототипирования распараллеливающего компилятора. Универсальное внутреннее представление системы 212
Мельников Л. С., Петренко И. В. Путевые ядра и длины циклов в неориентированных графах 222
Мехонтцев Д. Ю., Лобив И. В., Селезнев К. С. Слежение и определение скорости движущихся на плоскости объектов в реальном времени 232
Терехов В. И., Треногин Н. Г. Оптимальное размещение информации в вычислительных системах с учетом структурной надежности компонентов 243

CONTENTS

Preface 5
Baburin D. E. Hierarchical approach for drawing directed graphs 7
Volyanskaya T. A. Methods and techniques of adaptive hypermedia 38
Gluhankov M. P., Dortman P. A., Pavlov A. A., Stasenko A. P. Translating components of a functional programming system (SFP) 69
Dunaev A. A., Lobiv I. V., Mehontsev D. Yu., Murzin F. A., Polovinko O. N., Semich D. F., Chepel' A. V., Yarkov K. A. Algorithms of fast search for fragments of photoimages 88
Dylykov J. L.-D., Murzin F. A. TRIZ_Computing system 110
Dylykov J. L.-D., Pustyl'nikov V. A. Computer-aided decision making at the railway transport as a guarantee of railway safety 120
Evstigneev V. A. A pipeline model of transportation as a model of message passing 129
Evstigneev V. A., Mirzuitova I. L. NUMA-architectures: current state 139
Ivanov M. A. A review of MPEG-like methods of video data encoding 155
Kasyanov V. N., Nesgovorova G. P., Volyanskaya T. A. Virtual museum of informatics history in Siberia 169
Kryazhenkov P. B. User interface of an integrated functional programming environment SFP 182
Malinina Yu. V. IS TRANSFORM: computer-aided system filling 201
Markin V. A., Markina S. A. A project of a system for fast prototyping of a parallelizing compiler. General-purpose internal system representation 212
Mel'nikov L. S., Petrenko I. V. Path kernel and cycle length in undirected graphs 222
Mekhontsev D. Yu., Lobiv I. V., Seleznev K. S. Real-time tracking and velocity determination of objects moving on a plane 232
Terekhov V. I., Trenogin N. G. Optimal data distribution in information systems according to the system component reliability 243

УДК 519.68 + 681.3.06

Иерархический подход для автоматического размещения ациклических графов / Бабурин Д. Е. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 7–37.

Данная статья представляет собой обзор методологии для конструирования геометрического изображения ациклических графов. Для подчеркивания иерархичности структуры таких графов используются поуровневые представления, в которых все дуги следуют одному и тому же направлению. Иерархический подход содержит три основные части.

1. Распределение вершин по уровням так, чтобы все дуги следовали одному направлению.

2. Выбор порядка вершин на уровне с целью минимизации пересечения ребер.

3. Определение координат вершин на уровне с целью минимизации общей длины ребер и количества изломов.

Обзор содержит детальное описание каждой из частей алгоритма. Автор приводит различные способы решения задач того или иного этапа с указанием первоисточников, а также проводит сравнение этих решений по удовлетворяемым эстетическим критериям и временной сложности. Помимо решения перечисленных задач, в обзоре затронуты смежные вопросы (расстановка меток, инкрементальные размещения, открытые проблемы) и описан ряд методов для сведения произвольного графа к ациклическому. — Библиогр.: 78 назв.

Hierarchical approach for drawing directed graphs / Baburin D. E. // Modern problems of program construction. — Novosibirsk, 2002. — P. 9–30.

This article is a review of the methodology for automatic drawing of arbitrary directed graphs. Layered drawings with edges following one common direction are used to emphasize the hierarchy of vertices in such graphs. The approach described in this article is usually divided into three steps.

1. Assignment of the vertices to layers so that all edges point to one direction.

2. For each layer, an ordering of the vertices is computed in order to minimize the number of edge crossings.

3. Coordinate assignment of vertices in the layer in order to minimize the total edge length and the number of edge bends.

The review contains a detailed description of each step of the algorithm. The author presents different solutions to the problems arising at each stage, gives references to original papers, and also compares these solutions in terms of satisfying aesthetic criteria and time complexity. The review also covers the related topics (edge and vertex labeling, dynamic layouts, open problems) and a number of methods to convert an arbitrary graph to a directed one. — Refs: 78 titles.

УДК 519.68 + 681.3.06

Методы и технологии адаптивной гипермедиа / Волянская Т. А. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 38–68.

Адаптивная гипермедиа — относительно новое направление исследований на стыке гипермедиа и моделирования пользователя. Адаптивные гипермедиа-системы формируют модель пользователя из целей, знаний, предпочтений и других характеристик индивидуального пользователя и используют ее в течение взаимодействия для адаптации к потребностям этого пользователя. Статья содержит краткий обзор существующих на данный момент методов и технологий адаптивного представления содержания и адаптивной навигационной поддержки. — Библиогр.: 6 назв.

Methods and techniques of adaptive hypermedia / Volyanskaya T. A. // Modern problems of program construction. — Novosibirsk, 2002. — P. 38–68.

Adaptive hypermedia is a relatively new research direction at the junction of hypermedia and user modeling. Adaptive hypermedia systems build a user model on the basis of goals, knowledge, preferences, and other characteristics of each particular user, and use this model when interacting with the user in order to adapt to his/her needs. This paper provides a brief survey of existing methods and techniques of adaptive content presentation and adaptive navigation support. — Refs: 6 titles.

УДК 519.68 + 681.3.06

Транслирующие компоненты системы функционального программирования SFP / Глуханков М. П., Дортман П. А., Павлов А. А., Стасенко А. П. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 69–87.

В данной работе описывается система функционального программирования SFP и ее компоненты: транслятор, средства отладки, внутреннее представление Sisal-программ, конвертер Sisal-программ в программы на языке Си. Также рассматривается тестирование транслирующих компонент системы. — Библиогр.: 34 назв.

Translating components of a functional programming system (SFP) / Gluhankov M. P., Dortman P. A., Pavlov A. A., Stasenko A. P. // Modern problems of program construction. — Novosibirsk, 2002. — P. 69–87.

The paper describes a system of functional programming (SFP) and its components: translator, debugging tools, internal representation of Sisal-programs, Sisal-to-C converter. Testing of SFP's translating components is also described. — Refs: 34 titles.

УДК 519.68 + 681.3.06

Алгоритмы быстрого поиска фрагментов фотографических изображений / Дунаев А. А., Лобив И. В., Мехонцев Д. Ю., Мурзин Ф. А., Половинко О. Н., Семич Д. Ф., Чепель А. В., Ярков К. А. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 88–109.

Поиск фрагментов в изображениях — важная задача, возникающая во многих отраслях человеческой деятельности. При этом поиск должен осуществляться максимально быстро.

Цель данной работы — создание алгоритмов и программ, обнаруживающих заданный фрагмент на изображении. Представленная программа состоит из двух частей — вычислительного ядра и управляющей оболочки, осуществляющей пользовательский интерфейс.

Главное преимущество нового подхода состоит в том, что все образцы разыскиваются одновременно. Разработанный алгоритм является одним из лучших в мире.

Algorithms of fast search for fragments of photoimages / Dunaev A. A., Lobiv I. V., Mehontsev D. Yu., Murzin F. A., Polovinko O. N., Semich D. F., Chepel' A. V., Yarkov K. A. // Modern problems of program construction. — Novosibirsk, 2002. — P. 88–109.

Search for fragments in images is an actual problem arising in many fields of human activity. It is desirable to perform it with maximal speed.

The aim of this work is to construct algorithms and programs that find a particular fragment in an image. The program here described consists of two parts: a computational kernel and a control shell that works as a user interface.

The main advantage of this new approach is the fact that all fragments are in a simultaneous search. This algorithm is one of the best in the world.

УДК 519.68 + 681.3.06

Система TRIZ_Computing / Дылыков Ж. Л-Д., Мурзин Ф. А. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 110–119.

В статье исследованы методы принятия решения в теории решения изобретательских задач (ТРИЗ) и их применение к проблеме разрешения конфликтов. Рассмотрен алгоритм АРИЗ и его аналог для решения конфликтных ситуаций АРПС. Описана графическая оболочка TRIZ_Computing для разбора конфликта. В программе разобраны несколько примеров. — Библиогр.: 11 назв.

TRIZ_Computing system / Dylykov J. L.-D., Murzin F. A. // Modern problems of program construction. — Novosibirsk, 2002. — P. 110–119.

This article is devoted to the methods of decision making in the theory for solving the inventive problems (TRIZ) and their application to the problem of conflict resolution. The algorithm of solving the inventive problems (ARIZ) and its analogue for resolution of conflict situations (ARPS) are considered. A graphical interface of TRIZ_Computing intended for conflict analysis is described. The program presents several examples. — Refs: 11 titles.

УДК 519.68 + 681.3.06

Автоматизация методов принятия решений на железнодорожном транспорте как гарантия обеспечения безопасности движения / Дылыков Ж. Л-Д., Пустыльников В. А. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 120–128.

В статье представлен комплекс АРМов цеха эксплуатации, называемых АРМ ТЧБ (Арм Нарядчика, Арм Зав.Бригадами). АРМ ТЧБ разработан в рамках концепции автоматизированной системы управления локомотивным хозяйством (АСУТ), разработанной по заданию Департамента локомотивного хозяйства в Отраслевом Центре Внедрения МПС совместно с ВНИИЖТ, ВНИИАС, Красноярской и Московской железными дорогами и другими предприятиями железнодорожного транспорта.

Основное назначение системы — полностью исключить ошибки оперативного персонала при формировании и использовании локомотивных бригад и

машинистов, работающих в одно лицо (только машинист, без помощника на специально оборудованном локомотиве). Система также предназначена для оптимизации процесса управления работой локомотивных бригад. Еще одной не менее важной задачей АРМ ТЧБ (в комплексе с АРМ ТЧД) является автоматизация подготовки и отправки в систему оперативного контроля за дислокацией бригад (ОКДБ) сообщений о дислокации и состояниях бригад в реальном масштабе времени. — Библиогр.: 5 назв.

Computer-aided decision making at the railway transport as a guarantee of railway safety / Dylykov J. L.-D., Pustyl'nikov V. A. // Modern problems of program construction. — Novosibirsk, 2002. — P. 120–128.

This article presents ARMs TCHB systems (ARM Naradchick, ARM ZAVbrigadami). ARM TCHB is implemented within the framework of ASUT (locomotive facilities automated control system) concept developed in the Application Center of Ministry of Railways in collaboration with VNIIZhT, VNIIAS, West-Siberian and Moscow railways and other railway enterprises by instruction of the Department of locomotive facilities.

The main purpose of the system is to completely exclude mistakes of operating personnel during formation and work of locomotive brigades and engine drivers working as one person. The system is also intended for optimization of managing the locomotive brigades. Another task (which is not of less importance) of ARM TCHB (conjugate with ARM TCHD) is computer-aided preparating and sending reports on disposition and state of brigades to OKDB system in a real-time mode. — Refs: 5 titles.

УДК 519.68 + 681.3.06

Конвейерная модель перевозок как модель пересылки протяженных сообщений / Евстигнеев В. А. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 129–138.

В данной работе показано, что наиболее близкой к задаче о “червеобразной” модели пересылки сообщений в мультипроцессорных системах является сетевая транспортная задача о перевозках однородного груза на основе конвейерной модели. — Библиогр.: 7 назв.

A pipeline model of transportation as a model of message passing / Evstigneev V. A. // Modern problems of program construction. — Novosibirsk, 2002. — P. 129–138.

We show that the transportation problem with a pipeline model is the closest to the problem with a “wormlike” model of message passing. — Refs: 7 titles.

УДК 519.6 + 681.3.06

Развитие NUMA-архитектуры: текущее состояние / Евстигнеев В. А., Мирзуитова И. Л. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 139–154.

Цель этой статьи — рассмотреть динамику развития NUMA-архитектур за последние 10 лет. — Библиогр.: 2 назв.

The development of NUMA-architectures: current state / Evstigneev V. A., Mirzuitova I. L. // Modern problems of program construction. — Novosibirsk, 2002. — P. 139–154.

The aim of this paper is to consider the dynamic of development of NUMA-architectures in the last decade. — Refs: 2 titles.

УДК 519.68 + 681.3.06

Обзор MPEG-подобных методов кодирования видеоданных / Иванов М. А. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 155–168.

Рассмотрены основные методы кодирования видеоданных, использующиеся в MPEG-подобных форматах: поиск векторов движения, кодирование остаточного изображения, сжатие данных специального вида. — Библиогр.: 5 назв.

A review of MPEG-like methods of video data encoding / Ivanov M. A. // Modern problems of program construction. — Novosibirsk, 2002. — P. 55–168.

The basic methods of video data encoding used in MPEG-like formats are considered, such as a search for motion vectors, encoding of residual image, and encoding of special data. — Refs.: 5 titles.

УДК 519.68 + 681.3.06

Виртуальный музей истории информатики в Сибири / Касьянов В. Н., Несговорова Г. П., Волянская Т. А. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 169–181.

В статье представлен проект создания виртуального музея истории информатики в Сибири. Кратко описана сибирская школа информатики и программирования, рассматривается структура виртуального музея и его содержимое, описывается пользовательский интерфейс и изучаются вопросы адаптивной гипермедиа. — Библиогр.: 3 назв.

Virtual museum of informatics history in Siberia / Kasyanov V. N., Nesgovorova G. P., Volyanskaya T. A. // Modern problems of program construction. — Novosibirsk, 2002. — P. 169–181.

The paper presents the project of a virtual museum of informatics history in Siberia. The Siberian informatics and programming school is briefly described, the structure, contents and user interface of the virtual museum are considered, and adaptive hypermedia problems are studied. — Refs: 3 titles.

УДК 519.68 + 681.3.06

Пользовательский интерфейс интегрированной среды функционального программирования SFP / Кряженков П. Б. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 182–200.

Публикация посвящена работе, которая является частью проекта по созданию интегрированной среды программирования для функционального языка Sisal. В статье описан интерфейс к интегрированной среде программирования функционального языка Sisal, а также некоторые компоненты, такие как редактор, транслятор. — Библиогр.: 8 назв.

User interface of an integrated functional programming environment SFP / Kryazhenkov P. B. // Modern problems of program construction. — Novosibirsk, 2002. — P. 182–200.

The paper is devoted to a part of a project on creation of IFPE (integrated functional programming environment) for the functional programming language Sisal. It describes the user interface of IFPE and some of its components, such as editor, translator, etc. — Refs: 8 titles.

УДК 681.3.016-681.3.06

ИС ТРАНСФОРМ: автоматизация наполнения системы / Малинина Ю. В. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 201–211.

Статья посвящена описанию результатов опытной эксплуатации ИС ТРАНСФОРМ и возможностям ее дальнейшего развития. Так же рассматриваются существующие методы индексации и возможные подходы к решению проблемы адекватного автоматического индексирования документов и извлечения из них сопутствующей информации. — Библиогр.: 11 назв.

IS TRANSFORM: computer-aided system filling / Malinina Yu. V. // Modern problems of program construction. — Novosibirsk, 2002. — P. 201–211.

This paper focuses on the current state of IS TRANSFORM and possible ways of extending its features. It also considers the methods of autoindexing and possible approaches to the decision of the problem of adequate autoindexing of documents and extracting information from them. — Refs: 11 titles.

УДК 519.68 + 681.3.06

Проект системы для быстрого прототипирования распараллеливающего компилятора. Универсальное внутреннее представление системы / Маркин В. А., Маркина С. А. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 212–221.

В первой части статьи представлен проект исследовательско-обучающей системы для быстрого создания прототипа распараллеливающего компилятора. Приведены поставленные перед системой задачи и их связь с процессом создания компилятора. Дано описание архитектуры создаваемой системы, ее составных частей, относящихся к определенным этапам работы транслирующей системы и основных компонентов, таких как расширяемое универсальное внутреннее представление (ВП), сценарий и язык сценария системы. Во второй части статьи детально рассмотрено ВП системы и его языковая и графовая составляющие. Введены понятия абстрактного синтаксического графа и графовых теней для исследования различных теоретико-графовых внутренних представлений программ. — Библиогр.: 16 назв.

A project of a system for fast prototyping of a parallelizing compiler. General-purpose internal system representation / Markin V. A., Markina S. A. // Modern problems of program construction. — Novosibirsk, 2002. — P. 212–221.

The first part of this paper presents a project of a research-and-training system for fast prototyping of the parallelizing compiler. It contains the tasks set to the system and shows how they relate to the process of compiler development. The architecture of the system is described together with its constituents working at certain stages of compilation and its main components, such as a scalable universal internal form (IF), “scenario” and the “scenario” language. In the second part of the paper, IF of the system, its “language” and “graph” components are considered in detail. Some new concepts are introduced, namely, the abstract syntactical graph and graph shadows for investigation of various internal graph-theoretical forms of programs. — Refs: 16 titles.

УДК 519.68 + 681.3.06

Путевые ядра и длины циклов в неориентированных графах / Мельников Л. С., Петренко И. В. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 222–231.

Через $\tau(G)$ обозначим количество вершин в наиболее длинном пути конечного неориентированного графа G. Подмножество K множества вершин V графа G называется P_n–ядром графа G, если $\tau(G[K]) \leq n-1$ и каждая вершина из множества $V(G \backslash K)$ смежна с вершиной, которая является конечной вершиной для пути длины n в графе G. В данной статье доказана теорема о существовании P_n–ядра в графе, длина максимального цикла в котором равна $n-2$. — Библиогр.: 14 назв.

Path kernel and cycle length in undirected graphs / Mel'nikov L. S., Petrenko I. V. // Modern problems of program construction. — Novosibirsk, 2002. — P. 222–231.

Let $\tau(G)$ denote the number of vertices in the longest path of a graph G. A subset K of the set of vertices V of the graph G is called P_n–kernel of the graph G, if $\tau(G[K]) \leq n-1$ and every vertex $x \in V(G \setminus K)$ are adjacent to the terminal vertex of the path of length n in the graph G. Here the theorem on existence of a P_n–kernel of the graph G with the length of the maximal cycle $n-2$ has been proved. — Refs: 14 titles.

УДК 519.68 + 681.3.06

Слежение и определение скорости движущихся на плоскости объектов в реальном времени / Мехонтцев Д. Ю., Лобив И. В., Селезнев К. С. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 243–246.

В данной статье описан программный логический модуль для системы отслеживания движения объектов в реальном времени. На вход поступают изображения, полученные с неподвижно закрепленной видеокамеры, на выходе имеются данные об объектах, которые попали в поле зрения. В качестве аппаратной части выступает обычный компьютер класса Pentium-2. — Библиогр.: 3 назв.

Real-time tracking and velocity determination of objects moving on a plane / Mekhontsev D. Yu., Lobiv I. V., Seleznev K. S. // Modern problems of program construction. — Novosibirsk, 2002. — P. 243–246.

A logical module is discussed in the paper for a real-time object tracking system. Images obtained by a fixed video camera are used as input data. The output is information about objects viewed by the camera. A usual Pentium-2 computer is used as the hardware part. — Refs: 3 titles.

УДК 519.68 + 681.3.06

Оптимальное размещение информации в вычислительных системах с учетом структурной надежности компонентов / Терехов В. И., Треногин Н. Г. // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 243–246.

Данная статья посвящена проблеме оптимального размещения данных в информационных системах. Рассматривается случай распределенной клиент-серверной системы. В статье предложена методика оптимального размещения таблиц по серверам баз данных. Предлагаемый подход учитывает не

только параметры быстродействия системы, пропускную способность каналов, но и надежность компонентов системы. Приведенная методика может быть использована при проектировании вычислительных систем. — Библиогр.: 1 назв.

Optimal data distribution in information systems according to the system component reliability / Terekhov V. I., Trenogin N. G. // Modern problems of program construction. — Novosibirsk, 2002. — P. 243–246.

The article is devoted to the problem of optimal data distribution in information systems for the case of a distributed client-server system. The method of optimal distribution of tables is described. The proposed approach considers not only the system's component performance and data channel utilization, but also the system reliability. It is possible to use this method in the computation system design. — Refs: 1 titles.

СОВРЕМЕННЫЕ ПРОБЛЕМЫ КОНСТРУИРОВАНИЯ ПРОГРАММ

Под редакцией
проф. Виктора Николаевича Касьянова

Рукопись поступила в редакцию 19. 04. 2001

Ответственный за выпуск Г. П. Несговорова
Редактор З. В. Скок

Подписано в печать 27. 09. 2002
Формат бумаги 60 × 84 1/16 Объем 14,6 уч.-изд.л., 16,0 п.л.
Тираж 75 экз.

НФ ООО ИПО "Эмари" РИЦ, 630090, г. Новосибирск, пр. Акад. Лаврентьева, 6