СОДЕРЖАНИЕ

Номер 6, 2020

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

ПРОГРАММНАЯ ИНЖЕНЕРИЯ, ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ ПРОГРАММ

ПАРАЛЛЕЛЬНОЕ И РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ

ТЕОРЕТИЧЕСКИЕ ВОПРОСЫ ПРОГРАММИРОВАНИЯ

CONTENTS

No. 6, 2020

ПРОГРАММИРОВАНИЕ, 2020, № 6, с. 3–15

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.42

РАСПОЗНАВАНИЕ РЕКВИЗИТОВ БАНКОВСКИХ КАРТ В МОБИЛЬНЫХ УСТРОЙСТВАХ ПО ВИДЕОПОСЛЕДОВАТЕЛЬНОСТЯМ

© 2020 г. Х. Чен[a,*], Ш. Е[a,], А. Курилович[b,***], Р. Богуш[b,****], С. Абламейко[c,d,*****]**

[a] *Университет Чжэцзян Шурэнь, Ханчжоу, ул. Шурэнь, 8, 310009 Китай*

[b] *Полоцкий государственный университет, Новополоцк, ул. Блохина, 29, 211440 Республика Беларусь*

[c] *Белорусский государственный университет, Минск, п-т Независимости, 4, 220030 Республика Беларусь*

[d] *Объединенный институт проблем информатики НАН Беларуси, Минск, ул. Сурганова, 6, 220012 Республика Беларусь*

**E-mail: eric.hf.chen@hotmail.com*

***E-mail: zjsruysp@163.com*

****E-mail: cfif921@yandex.ru*

*****E-mail: bogushr@mail.ru*

******E-mail: ablameyko@bsu.by*

Поступила в редакцию 08.10.2019 г.
После доработки 05.11.2019 г.
Принята к публикации 05.12.2019 г.

Предлагается алгоритм для обнаружения и распознавания всех информационных полей лицевой стороны банковских карт по видеопоследовательностям, который предназначен для использования в мобильных устройствах. Данный алгоритм включает следующие основные шаги: обнаружение границ карты в кадре, сегментация информационных полей, улучшение изображений сегментов, выделение границ символов и распознавание блоков символов. Доля истинно положительной классификации предложенным алгоритмом всех трех полей данных на карте составляет 0.88, для номера и срока действия карты – 0.925. Представлены временные затраты на обработку видеопоследовательностей с различным разрешением кадра с применением iPhone7. Результаты экспериментов подтверждают эффективность предложенного подхода.

DOI: 10.31857/S0132347420060023

1. ВВЕДЕНИЕ

Развитие банковских технологий привело к широкому внедрению интернет-банкинга, т.е. дистанционного банковского обслуживания клиентов через сеть Интернет, а также мобильного банкинга, который предоставляет возможность управления банковскими счетами с помощью планшетного компьютера или смартфона. В настоящее время у одного клиента банка в наличии может находиться две и более банковские карты, которые он может использовать для оплаты. Заполнение всех полей данных при оплате услуг или осуществлении покупок с использованием мобильного банкинга требует временных затрат и внимательности. Поэтому, совершенствование мобильного банкинга предполагает разработку алгоритмического и программного обеспечения для автоматизации ввода реквизитов банковской карты в систему на основе анализа ее изображения, полученного с использованием мобильного устройства.

Дизайн банковской карты является важным носителем бренда банка, что не накладывает практически никаких ограничений на яркостно-цветовые характеристики при его разработке. Таким образом, изображение на карте может быть неоднородным и достаточно сложным. Кроме того, пластик с глянцевой поверхностью, из которого, как правило, изготавливаются банковские карты, обладает сильными отражающими характеристиками, что при ярком освещении приводит к возникновению бликов и отражений на полу-

ченных изображениях, особенно при использовании вспышки в условиях недостаточной освещенности. При наличии теней или недостаточном освещении изображение карты может оказаться достаточно низкого качества.

Банковская карта является типичным примером документа с гибкой формой, поэтому при ее обработке могут быть использованы алгоритмы обнаружения и распознавания данных, которые используются на других типах документов с гибкой формой [1]. В работе [2] представлен подход для сегментации и распознавания текстовых символов на визитных картах с однотонным фоном. При этом используются такие основные шаги, как выделение краев оператором Собеля, утоньшение линий, проективное преобразование, адаптивная бинаризация, сегментация слов и текстовых символов с последующим их распознаванием. Применение данного подхода ограничивается возможностью работы только с простым фоном и достаточно значительными вычислительными затратами, что накладывает ограничение на его реализацию в мобильных устройствах. В работе [3] предложен алгоритм обработки визитных карт на первом этапе которого выполняется грубое удаление фона на основе блочного анализа входного изображения, на втором этапе классифицируются полученные связные компоненты для определения текстовых регионов и исключения изображений логотипов. Затем к результату применяется адаптивная бинаризация, которая позволяет отделить текст от фона для дальнейшего распознавания. Данный алгоритм также может быть использован только для изображений карт с монотонным фоном. Кроме этого, не предусмотрено проективное преобразование, что накладывает дополнительные требования к ориентации входного изображения относительно камеры смартфона.

Методы распознавания данных визитных карты для платформы Андроид представлены в [4] и [5]. В [4] используется программный пакет MatLab для предварительной обработки и сегментации текстовых символов, и библиотека Tesseract для их распознавания. Однако, как отмечают авторы, предварительная обработка используется ряд операций с большой вычислительной сложностью по улучшению входного изображения и сегментации символов, поэтому на мобильном устройстве реализован упрощенный вариант алгоритма с отсутствием определенных этапов предварительной обработки, что привело к значительному снижению качества его работы. На первом шаге метода из [5] выполняется уменьшение размера входного изображения, операции предобработки выполняются с использованием библиотеки OpenCV, распознавание символов осуществляется на основе Tesseract. Однако результаты экспериментов в работах представлены для изображений визитных карт с однотонным фоном, на котором хорошо видны текстовые символы.

Алгоритм распознавания только даты окончания действия кредитной карты и оценка его качественных характеристик с использованием смартфона iPhone 4S представлен в [6]. Его особенностью является то, что он не обеспечивает автоматизации ввода всех данных с изображения и, соответственно, требуется дальнейшее его развитие. Возможности алгоритмов распознавания номеров банковских карт для мобильных устройств под управлением операционной системы Android, рассмотренных в [7] и [8], также не включает распознавание имени и фамилии пользователя. В [7] точность распознавания достигает 80%, а в [8] использование предварительной обработки изображений карт и рекуррентной нейронной сети обеспечивает точность распознавания цифр до 90%. Следует также отметить, что данные алгоритмы предназначены для работы с эмбоссированными картами, у которых на лицевой стороне цифры и буквы выдавлены. В [9] приведен метод построения функции надежности распознавания изображений тисненых символов и показана его эффективность для повышения вероятности их правильного распознавания. Однако, широкое распространение получили неэмбоссированные карты, которые, могут выдаваться клиенту в день его обращения в банк. Но из-за отсутствия процедуры выдавливания буквенно-цифровой информации ее локализация и распознавание значительно сложнее и требует эффективной предварительной обработки для локализации символов и обеспечения возможности их правильного распознавания.

Существующие подходы используют в качестве входных данных статические изображения, на которых в ряде случаев могут возникать шумы, блики, значительные перепады яркости, иметь недостаточною резкость и т.д., что приводит к необходимости получения нового изображения, которое также может быть непригодно для правильного обнаружения карты, информационных полей на ней и (или) дальнейшего их распознавания. Повышение результативности обработки данных может быть обеспечено за счет использования видео, которое легко получается с помощью мобильного телефона. Однако, наличие не одного изображения, а их последовательности приводит к необходимости разработки других подходов к обработке. В работе [10] было предложено использовать видеопоследовательности для обнаружения краев документов мобильными устройствами. При этом, сегментация изображений выполняется в цветовом пространстве Lab с использованием морфологической обработки, связывание границ

отдельных сегментов на основе преобразования Хафа и обнаружение линий, ограничивающих документ в целом. Однако, для сокращения вычислительных затрат применяется уменьшение размера кадра входного изображения с 1280 × 720 до 180 × 100 пикселей. Такое уменьшение кадра не пригодно для решения задачи локализации и распознавания информации на банковской карте, так как вероятность распознавания символов будет низкой из-за их малых размеров.

В данной работе предлагается алгоритм для обнаружения и распознавания реквизитов банковских карт по видеопоследовательностям в мобильных устройствах. Алгоритм позволяет практически в реальном времени распознавать все информационные поля лицевой стороны банковской карты, включая номер, дату окончания действия, имя и фамилию владельца на латинице и на кириллице, для эмбоссированных и неэмбоссированных типов карт.

2. АЛГОРИТМ РАСПОЗНАВАНИЯ РЕКВИЗИТОВ БАНКОВСКИХ КАРТ ПО ВИДЕОПОСЛЕДОВАТЕЛЬНОСТЯМ

Разработанный алгоритм показан на рис. 1. Последовательность кадров, полученная с помощью видеокамеры мобильного устройства, поступает в блок детектирования прямоугольных областей, который обнаруживает все прямоугольники в кадре и возвращает только один, удовлетворяющий параметрам банковской карты.

Для данной области выполняется преобразование в полутоновое изображение, которое передается в блок сегментации для выделения и индексации областей на изображении карты, которые соответствуют таким информационным полям, как номер карты, срок ее действия, имя и фамилия владельца. Для сегментированных блоков выполняется улучшение контраста фильтрации. После этого применяется адаптивная бинаризация и морфологическая обработка. Следующий шаг предназначен для уточнения границ областей с символами и использует метод скользящего окна. После этого, сегментированные области передаются в блок OCR для распознавания цифровых и текстовых символов. В завершение, блок оценки данных обрабатывает полученную информацию для отображения либо отклонения результата. Если полученные номер карты и дата срока ее окончания действия являются валидными, результат для них считается правильным. Распознавание данных поля имени владельца прекращается после первого полученного результата. Если на текущем кадре считана информация не со всех трех информационных полей, то на следующем кадре обрабатываются только недостающие.

2.1. Обнаружение карты

При разработке алгоритма обнаружения карты учитывается, что размер кадра входной видеопоследовательности в современных мобильных устройствах варьируется и может достигать разре-

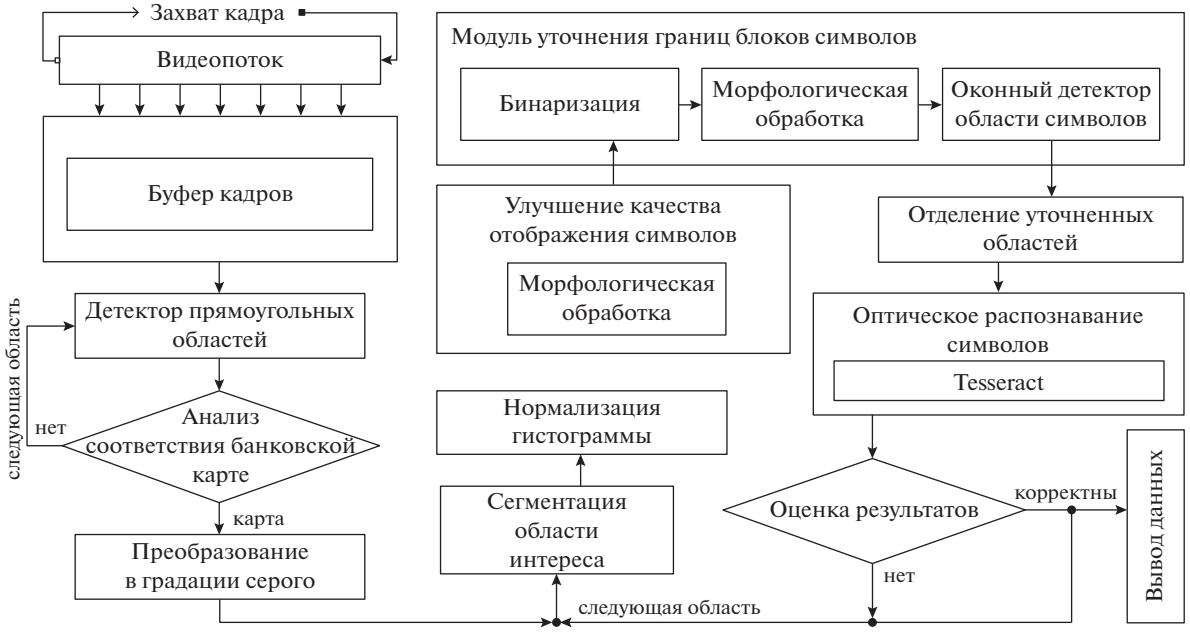


Рис. 1. Обобщенная схема алгоритма.

шения 3840 × 2160 пикселей, также устройствами обеспечивается автоматическое удержание фокуса на объекте, корректировка яркости и баланс белого. Таким образом, полученное изображение будет, как правило, иметь приемлемые следующие характеристики: резкость, яркость и достаточное разрешение кадра для автоматического извлечения данных банковских карт.

На первом этапе осуществляется детектирование и локализация всех прямоугольных областей на изображении. С учетом возможного большого разрешения кадра, до 4 K, но ограниченных вычислительных ресурсов мобильных устройств, применяется метод поиска, основанный на быстрых алгоритмах Виолы–Джонса [11] и OverFeat [12], обеспечивающий также высокую результативность обнаружения. Далее, для определения контура, описывающего банковскую карту, используется анализ отношения размеров сторон полученных объектов. Поскольку высота m_o и ширина n_o карты стандартизированы, отношение размеров ее сторон является постоянной величиной, то целесообразно ее использовать в качестве критерия при нахождении контура карты из всех детектированных прямоугольных областей. Тогда прямоугольная область r_i с размерами $m_i \times n_i$ может быть отнесена к изображению карты в кадре, если выполняется условие:

$$\frac{n_o}{m_o} - e < \frac{n_i}{m_i} < \frac{n_o}{m_o} + e, \qquad (2.1)$$

где e – коэффициент допустимого отклонения соотношения размеров сторон карты.

Однозначным соответствием считается прямоугольник с наибольшей длинной стороны m_{max} из всех детектированных. Найденные на данном этапе размеры прямоугольной области и ее расположение относительно всего изображения $p_i(x_i, y_i)$ используются для извлечения области карты из входного кадра.

2.2. Сегментация областей интереса

На полученном изображении карты I с размерами $m_I \times n_I$ необходимо найти области, содержащие информацию о номере банковской карты (I_C), дате истечения срока ее действия (I_D) и владельце (I_E). Размер и расположение этих областей определено ISO/IEC 7811-5.4:2018, что может быть использовано для сегментации этих полей, формально представлено как:

$C(x_C, y_C, m_C, n_C)$ – номер карты;

$D(x_D, y_D, m_D, n_D)$ – дата окончания срока действия карты;

$E(x_E, y_E, m_E, n_E)$ – имя и фамилия держателя карты.

Поскольку значения m_I и n_I могут варьироваться в процессе видеосъемки, необходимо предусмотреть нормализацию, чтобы корректно определять C, D и E. Для этого используются коэффициенты масштабирования, которые умножаются на параметры этих областей, по ширине ($c_{мш}$) и по высоте ($c_{мв}$):

$$c_{мш} = \frac{n_I}{n_o}, \quad c_{мв} = \frac{m_I}{m_o}. \qquad (2.2)$$

Номер банковской карты содержит 16 цифр, распределенных на 4 равные группы по 4 цифры в каждой. Исходя из этого выделенное на предыдущем этапе изображение I_C с областью номера карты C_I, разделяется на 4 равные области C_{I1}, C_{I2}, C_{I3}, C_{I4}:

$$C_{Ij}\left(x_{CI} + \left(\frac{n_{CI}}{4} \times (j-1)\right), y_{CI}, \frac{n_{CI}}{4}, m_{CI}\right), \quad j = 1,2,3,4. \qquad (2.3)$$

Выделив рассчитанные области из I_C, получим изображения сегментов групп номера карты – I_{C1}, I_{C2}, I_{C3}, I_{C4}.

2.3. Улучшение изображения сегментов

Для уменьшения количества обрабатываемой информации полученные изображения (I_D, I_E, $I_{CI} - I_{C4}$) преобразуются в полутоновые с дальнейшим повышением их контраста методом нормализации гистограммы, который обеспечивает растяжку не всего диапазона изменения интенсивностей, а только его наиболее информативного участка, что усиливает эффект контрастности за счет потери шумовых областей с редко встречающимися интенсивностями.

При таком подходе выходной уровень яркости g для пикселя определяется следующим образом:

$$g_{x,y} = \frac{255 \cdot (Y_{x,y} - Y_{c_{\min}})}{Y_{c_{\max}} - Y_{c_{\min}}}, \qquad (2.4)$$

где $Y_{x,y}$ – уровень яркости входного пикселя с координатами x, y; Y_{c_min}, Y_{c_max} – заданные значения уровней пикселей входного изображения, минимальное и максимальное соответственно.

Для улучшения результатов обработки Y_{c_min}, Y_{c_max} отличны от минимального и максимального уровней яркости входного изображения, что позволяет пренебречь незначительным количеством пикселей по краям гистограммы.

Таким образом, значения Y_{c_min}, Y_{c_max} могут быть определены как:

$$Y_{c_{\min}}(i) = \begin{cases} Y_{c_{\min}}(i+1), & \text{если} \quad \sum_{j=0}^{i} f_t(Y_j) < T \\ Y_i, & \text{в противном случае} \end{cases};$$
$$Y_{c_{\max}}(i) = \begin{cases} Y_{c_{\max}}(i-1), & \text{если} \quad \sum_{j=255}^{i} f_t(Y_j) < T, \\ Y_i, & \text{в противном случае} \end{cases} \quad (2.5)$$

где f_t – количество пикселей на изображении с яркостью Y_j; $T = m \cdot n \cdot e$; e – коэффициент допустимого отклонения (%), определяемый экспериментально.

Далее, с использованием операций математической морфологии, осуществляется подчеркивание границ символов на полученном с предыдущего этапа полутоновом изображении. Для того чтобы определить цвет фона и символов рассчитывается среднее значение яркости. Если результат больше 127, то цвет фона светлый, а цвет символов темный, и наоборот.

Если цвет символов определен как светлый, для изображения выполняется морфологическое преобразование WhiteTopHat, которое вычитает разомкнутое изображение из исходного, что обеспечивает подчеркивание деталей контуров светлых символов. Преобразование BlackTopHat используется, чтобы подчеркнуть детали границ темных символов путем вычитания исходного изображения из замкнутого. Структурирующий элемент b для ядер обоих фильтров имеет прямоугольную форму и его размер $n_b \times m_b$ с учетом экспериментально определенных коэффициентов масштабирования рассчитывается как:

$$n_b = 0.06 n_I, \quad m_b = 3 n_b, \quad (2.6)$$

где n_I, ширина поступившего для преобразования изображения.

2.4. Локализация границ

Поскольку на изображениях банковских карт, несмотря на их небольшой размер, яркостно-контрастные характеристики фрагментов могут значительно отличаться, уровни яркостей пикселей участков фона, на которых расположены символы, также изменяются. С учетом данной особенности и необходимости сохранения границ символов, используется метод бинаризации изображений с адаптивным порогом на основе анализа локального региона согласно выражению:

$$bin_{x,y} = \begin{cases} 255, & \text{если} \quad Y_{x,y} > T(x,y) \\ 0, & \text{в противном случае} \end{cases}, \quad (2.7)$$

где $T(x, y)$ – адаптивный порог бинаризации, рассчитываемый для каждого пикселя как $T(x, y) = corr(I, G) - c$; $corr(I, G)$ – взаимная корреляция локального фрагмента изображения размером $k \times k$ с окном Гаусса; c – константа. Матрица коэффициентов окна Гаусса с размером $k \times k$ определяется как [13]:

$$G_i = \alpha \cdot e^{\frac{-\left(i-\frac{(k-1)}{2}\right)^2}{2\cdot\sigma^2}}, \quad (2.8)$$
$$\sigma = 0.3 \cdot ((k-1) \cdot 0.5 - 1) + 0.8,$$

где $i = 0 \ldots k-1$, α – коэффициент масштабирования, выбранный таким образом, что $\sum_i^{\square} G_i = 1$

Для уменьшения количества шумов и удаления из изображения неинформативных деталей, а также для грубого отделения символов от фона изображения применяются последовательно морфологические операции замыкания и эрозии.

Форма структурирующего элемента b размером $n_b \times n_b \left(n_b = \frac{w_{symb}}{3} \right)$ ядер обоих фильтров задана с учетом особенностей символов шрифта согласно ISO/IEC 7811-7.1:2018 в виде эллипса. Толщина символа w_{symb} не зависит от типа обрабатываемого фрагмента изображения карты и определена для OCR-B. Проекция значения ширины шрифта (мм) на размер в пикселях, относительно размера области банковской карты, является константой и рассчитывается на основе ее полной ширины:

$$w_{symb} = 0.004884 \cdot n_0. \quad (2.9)$$

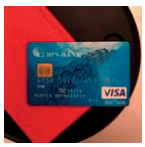
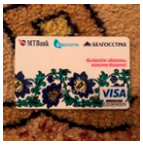
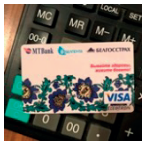
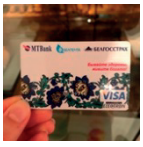
Далее выполняется уточнение краев области символов на обработанном изображении на базе метода скользящего вертикального окна. Высота окна варьируется по размеру шрифта символов. $H_{нк}$ – высота шрифта номера карты (4.0 мм), $H_{дси}$ – высота шрифта даты срока истечения карты (2.85 мм), $H_{ивк}$ – высота шрифта имени владельца карты (2.65 мм). Тогда их проекции на высоту в пикселях относительно размеров области полной карты $h_{нк}$, $h_{дси}$, $h_{ивк}$. Общий вид расчета представлен формулой:

$$h = \frac{H \cdot m_0}{54.0}, \quad (2.10)$$

где 54.0 мм – высота карты (ISO/IEC 7811-6.3:2018), а m_O – высота детектированной области карты (пикс).

Полученная область символов из предыдущего шага, черный и белый списки символов, а также

Таблица 1. Результаты тестирования алгоритма

Кадр из видеоряда			№	Карта обнаружена, (число кадров)	Число кадров с правильным распознаванием всех реквизитов/общее число кадров	Общее время, затраченное на обработку (мс)
1	2	3				
			1	22	9/90	1217
			2	–	–/90	–
			3	26	10/90	1221
			1	23	10/90	1284
			2	26	11/90	1149
			3	31	13/90	1265
			1	24	–/90	–
			2	21	7/90	1344
			3	–	–/90	–
			1	30	18/120	852
			2	37	21/120	924
			3	36	26/120	892
			1	39	–/135	–
			2	44	–/135	–
			3	41	–/135	–
			1	34	24/90	1324
			2	35	28/90	1238
			3	32	25/90	1287

языковой идентификатор последовательно передаются в систему распознавания Tesseract.

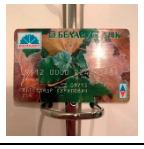
Белый список, состоящий из набора цифр от 0 до 9 и языковой идентификатор “eng”, передаются в качестве параметров для изображения номера банковской карты. Для изображения с датой срока ее истечения: цифры от 0 до 9 и символ “/” образуют белый список, а “eng” – идентификатор языка. Для имени держателя карты: “rus” и “eng” являются языковыми идентификаторами, а набор символов, включая знаки препинания, специальные символы и цифры, составляют черный список. Такой подход способствует увеличению скорости распознавания символов библиотекой Tesseract и ускорению всего процесса обработки в целом.

3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ И РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТОВ

3.1. Программная реализация

Программная реализация алгоритма выполнена на языке Objective-C с применением библиотеки низкоуровневой обработки изображений OpenCV, системы оптического распознавания символов Tesseract, фрэймворков iPhone SDK: CoreMedia и AV-

Таблица 2. Результаты сравнения подходов для распознавания информационных полей банковских карт

Исходные данные		CardIO			Предложенный алгоритм			
Изображение карты	Эмбоссинг символов	Время обработки при успешном распознавании	Результат распознавания информационного поля*		Время обработки при успешном распознавании	Результат распознавания информационного поля*		
			Номер карты	Срок действия		Номер карты	Срок действия	Держатель карты
	да	0.343	1	1	1.018	1	1	1
	нет	–	0	0	0.896	1	1	1
	да	0.314	1	1	1.059	1	1	0
	да	0.916	1	0	0	0	0	0
	нет	–	0	0	1.01	1	1	1
	да	0.864	1	1	0.971	1	1	1
	да	0.48	1	1	0.861	1	1	0
	нет	–	0	0	–	0	0	0
	да	0.478	1	0	0.957	1	1	1
	нет	–	0	0	0.986	1	1	1

* Примечание: 1 – соответствует успешному распознаванию информационного поля; 0 – соответствует неверному распознаванию.

Рис. 2. Внешний вид мобильного приложения, реализующего разработанный алгоритм: *1* – область просмотра изображения; *2* – метка успешного детектирования карты; *3* – область вывода распознанных данных.

Foundation – менеджмент медиаданных; UIKit – работа с интерфейсом приложения; CoreGraphics – низкоуровневая обработка 2D изображений на базе Quartz.

Фрэймворк AVFoundation используется для захвата видеопотока с основной камеры iOS-устройства в режиме реального времени с частотой 30 кадров в секунду. Кадры размещаются в очередях в отдельных потоках при помощи объектов класса NSOperationQueue. Первый поток получает кадры и размещает их в последовательной очереди для обработки, если последняя не занята, в противном случае кадры игнорируются. Второй поток последовательно извлекает кадры из очереди захвата и, передавая их на обработку, занимает очередь на период, пока не будут получены результаты. После завершения алгоритма, обработанный кадр удаляется, очередь освобождается, и итерация обработки запускается для следующего кадра.

Библиотека Tesseract выполняет обработку асинхронно в фоновом режиме. После окончания работы алгоритма полученный результат распознавания при помощи функции обратного вызова поступает для отображения пользователю в главном потоке. Интерфейс мобильного приложения с результатами распознавания показан на рис. 2.

3.2. Результаты экспериментов

Тестирование качественных характеристик разработанного алгоритма выполнено с использованием iPhone 7 и созданной базы данных из 180 банковских карт (рис. 3), которые размещались на сложном фоне, информационные поля реквизитов не были перекрыты посторонними предметами и визуально распознавались.

Значения коэффициентов, которые используются при вычислениях, были экспериментально получены с использованием методики ROC-анализа [14]. Вычисление допустимого коэффициента отклонения соотношения размеров сторон карты банковской карты (e) выполнено с учетом точных размеров ее согласно ISO/IEC 7811-5.4:2018 на выборке из 214 изображений, 178 из которых имели объекты карт, 36 – карт не имело, либо их границы были смазаны или искажены. В результате эксперимента для каждого коэффициента e, который изменялся от 0 до 0.02 с шагом в 0.0002, были рассчитаны доли истинного (TPR) и ложного (FPR) обнаружения. Для $e = 0.011$ получено максимальное значение TPR = 0.974194 и минимальное FPR = 0.111111.

Определение размера локального региона для бинаризации выполнено с учетом коэффициента масштабирования, так как данный параметр зависит от входных размеров детектированной области банковской карты (ее ширины (W_I, пикс.)):

$$k = W_I \cdot k'. \qquad (3.1)$$

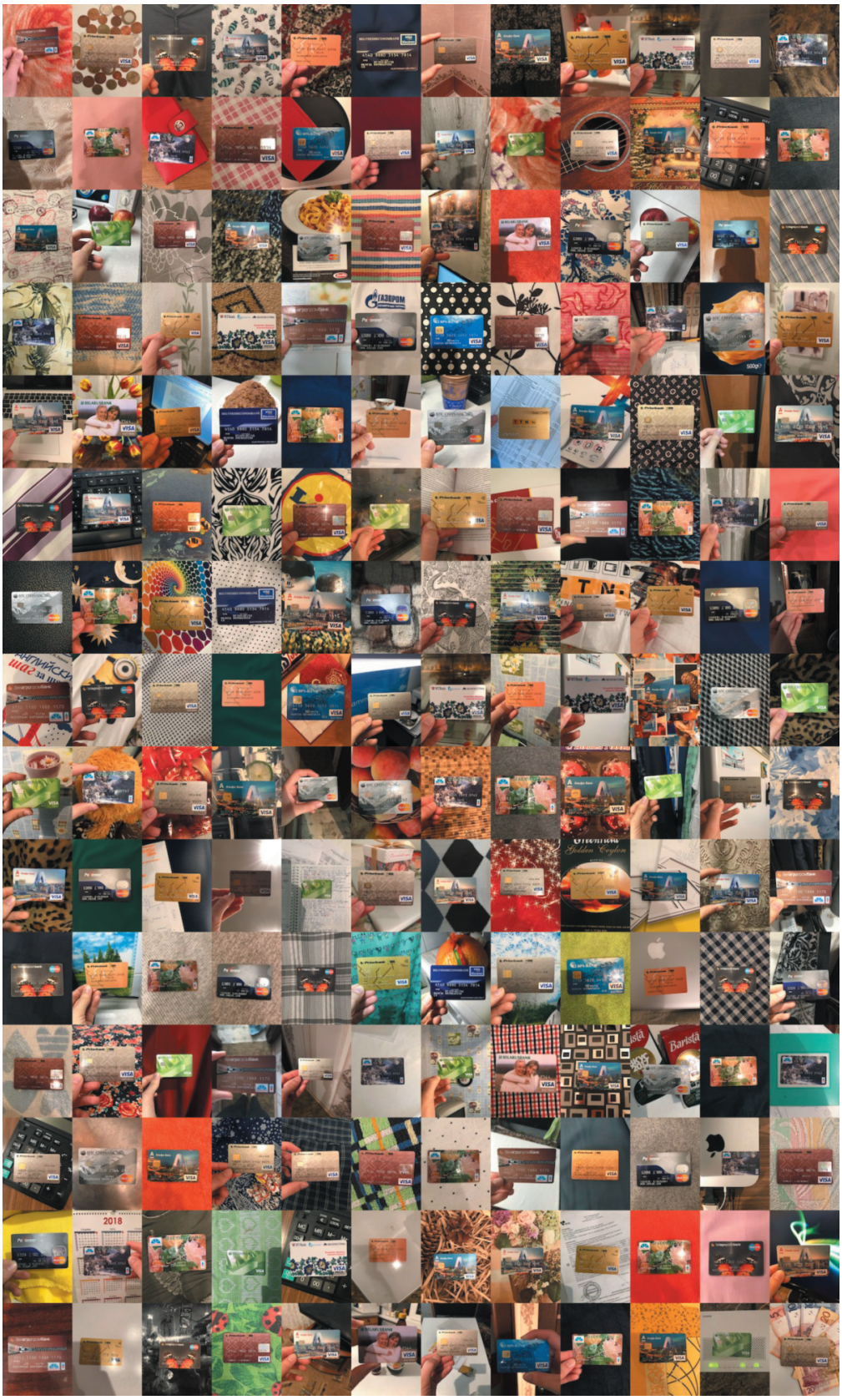


Рис. 3. Изображения используемых для тестирования банковских карт.

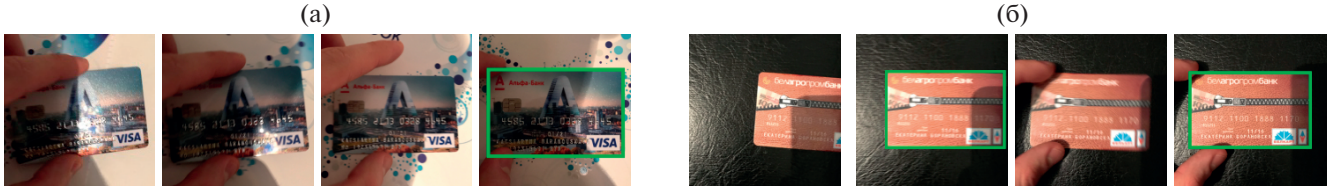


Рис. 4. Пример кадров видеопоследовательности.

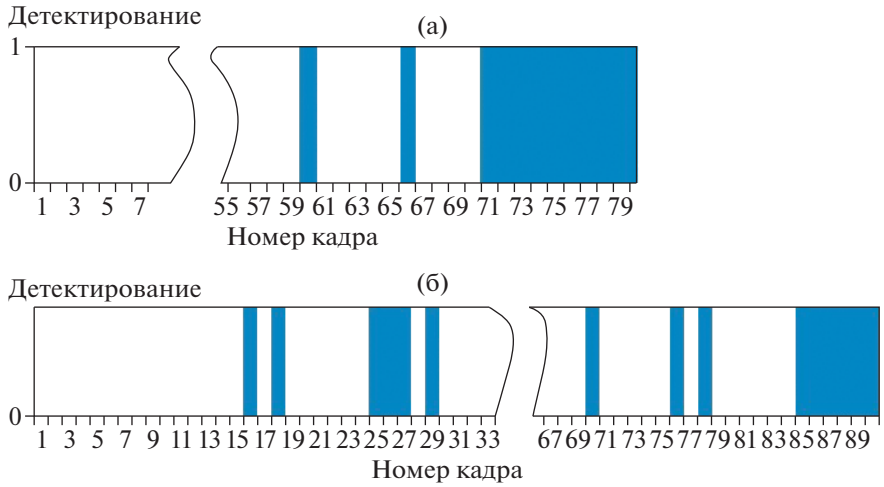


Рис. 5. Диаграмма детектирования карты на видеопоследовательности: а) для рис. 4а; б) для рис. 4б.

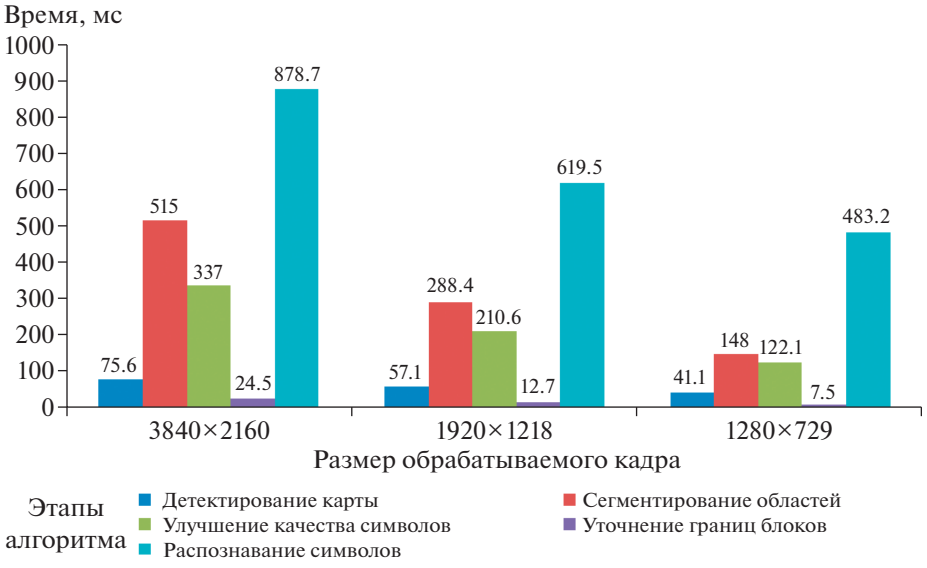


Рис. 6. Распределение временных затрат алгоритма.

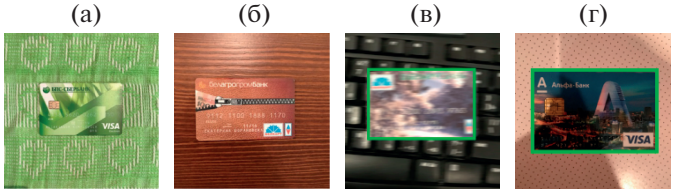


Рис. 7. Примеры изображений карт, которые не обнаруживаются или данные на них не распознаются.

Значения k' варьировались от 0.002 до 0.03 с шагом 0.001. На данном этапе максимальное значение доли истинного распознавания символов TPR = 0.83447 и доли ложного распознавания символов FPR = 0.140864 получено для $k' = 0.016$. Аналогично определена константа $c = 13$.

В табл. 1 представлены некоторые результаты выполненных исследований по оценке эффективности распознавания реквизитов карт на видеопоследовательностях, полученных в различных условиях, с применением разработанного подхода.

Согласно полученным с помощью ROC-анализа результатам, после установки оптимальных коэффициентов и пороговых значений в реализации алгоритма, доля истинно положительной классификации всех полей данных на карте TPR = 0.88, а доля ложноположительной классификации FPR = = 0.14. На рис. 4–6 приводятся примеры детектирования банковских карт из набора видеопоследовательностей, с указанием ключевых кадров и диаграмм детектирования. Обнаружению карты на диаграмме соответствует – 1, отсутствию – 0. Захват видео осуществлялся при постоянной частоте 30 кадр/с.

Анализ кадров видеопоследовательности, представленной на рис. 4а, свидетельствует, что изображение карты имеет сильные яркостные перепады, тени и блики, поэтому устойчивое ее обнаружение (начиная с 71-го кадра, рис. 5а) выполняется когда пользователь переносит карту в зону более стабильного освещения. На другой видеопоследовательности (рис. 4б) первоначально карта не имеет полных границ в кадре и не обнаруживается. Далее она успешно детектируется (рис. 5б), но имеет смазанный контур и нечеткие символы, что не позволит распознавать данные.

Временные затраты для каждого этапа обработки алгоритма одного кадра видеоряда различного разрешения с использованием устройства iPhone 7 показаны на рис. 6.

Как видно из графиков (рис. 6), время, затрачиваемое на операцию распознавания символов, имеет нелинейную зависимость от размера обрабатываемого кадра, в отличие от остальных операций.

3.3. Сравнение и обсуждение

Алгоритмы, предложенные в [7] и [8], характеризуются точностью распознавания цифр 80% и 90% соответственно. Однако, следует отметить, результаты получены на разных базах данных. Для обеспечения сравнения результатов работы представленного алгоритма с известными подходами в одинаковых условиях на одной и той же базе данных использовался реализованный модуль CardIO SDK [15].

Эксперименты выполнены для 180 карт при различных условиях сьемки, из которых для 104 случаев на картах присутствовало тиснение символов. В табл. 2 представлены примеры распознавания 10 изображений карт алгоритмом CardIO и предложенным алгоритмом.

Анализ результатов распознавания по всем картам используемой базы данных показывает, что разработанный подход значительно улучшает распознавание на картах с неэмбоссированными символами по сравнению с CardIO, который не предусматривает распознавание имени владельца карты. В общем, доля истинно положительной классификации предложенным алгоритмом всех трех полей данных на карте составляет TPR = 0.88, для номера и срока действия карты – TPR = 0.925. Модуль CardIO обеспечивает долю истинно положительной классификации цифровых полей

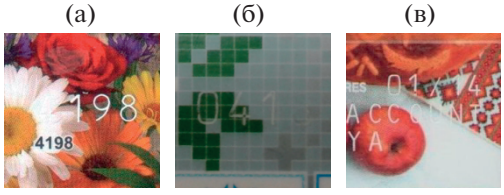


Рис. 8. Примеры фрагментов карт с поврежденными символами.

TPR = 0.68. Однако, только для карт с тиснением используемой базы данных CardIO характеризуется TPR = 0.935.

Если на входе разработанного алгоритма будут яркостно-цветовые характеристики изображения банковской карты очень схожи с общим фоном (рис. 7а, 7б), входное изображение размыто (рис. 7в), значительная область покрыта тенью, в результате чего цвет шрифта полностью совпадает с его фоном (рис. 10г), карты не могут быть им детектированы (рис. 7а, 7б) или данные на них не будут распознаны (рис. 7в, 7г). Кроме этого, невыразительный, не создающий контрастности с фоном карты контур символов, а также его повреждение или дефективность приводят к невозможности распознавания данных (см. рис. 8).

Анализ полученных результатов при проведении исследований показывает, что ошибки детектирования и распознавания возникают из-за влияния совокупности факторов: расположение объекта карты на сложном фоне, схожем с изображением самой карты; наличия бликов искусственного освещения или существенный перепад яркости; слишком удаленное ее расположение относительно камеры; смазывание контуров карты если камера или карта движется при видеосъемке. Дальнейшее улучшение работы алгоритма планируется за счет совершенствования алгоритма распознавания символов.

4. ЗАКЛЮЧЕНИЕ

Распознавание информационных полей банковских карт с использованием мобильных устройств является актуальной задачей. В данной статье представлен алгоритм для распознавания реквизитов лицевой стороны банковских карт с использованием мобильных устройств. Повышение результативности обнаружения карты и распознавания данных по сравнению с существующими подходами обеспечивается за счет анализа последовательности изображений, получаемой из видеоряда. Предложенный подход включает обнаружение границ карты в кадре на основе алгоритма Виолы–Джонса и метода OverFeat, сегментацию информационных полей с учетом их расположения на карте, улучшение изображений сегментов с применением нормализации гистограммы и морфологической обработки, определение границ блоков символов на основе адаптивной бинаризации и морфологической обработки, их уточнение с использованием скользящего вертикального окна, распознавание символов с применением библиотеки Tesseract. Программная реализация разработанного алгоритма выполнена с применением iPhone SDK, библиотек OpenCV и Tesseract. Тестирование предложенного алгоритма проведено с использованием мобильного устройства iPhone 7. При этом, доля истинно положительной классификации предложенным алгоритмом всех трех полей данных на карте составляет TPR = 0.88, для номера и срока действия карты – TPR = 0.925.

СПИСОК ЛИТЕРАТУРЫ

1. *Sheshkus A., Nikolaev D., Ingacheva A., Skoryukina N.* Approach to the recognition of flexible forms on the example of the credit card date recognition. Proceedings SPIE 9875, Eighth International Conference on Machine Vision (ICMV 2015), Barcelona, Spain, 8 December 2015, session 3. P. 83–88.
2. *Christian T., Gustavsson D.* Content recognition of business cards. Summer Project, IT University of Copenhagen, Copenhagen, Denmark, 5 September 2007.
3. *Mollah A., Basu S., Das N., Sarkar R., Nasipuri M., Kundu M.* Text Region Extraction from Business Card Images for Mobile Devices. Proceedings of International Conference on Information Technology and Business Intelligence (ITBI-09), Nagpur, India, 6–8 November 2009. P. 227–235.
4. *Bhaskar S., Lavassar N., Green S.* Implementing on the Android Operating System for Business Cards. IEEE 2010, EE 368 Digital Image Processing, Stanford, CA 7 June 2010. P. 1–5.
5. *Sharma P., Fujii K.* Automatic Contact Importer from Business Cards for Android. Stanford University, Stanford, CA, 2013.
6. *Hua G., Liu Z., Zhang Z., Wu Y.* Automatic business card scanning with a camera. International Conference on Image Processing, Atlanta, GA, USA 8–11 October 2006. P. 373–376.
7. *Cai S., Wen J., Xu H., Chen S., Ming Z.* Bank Card and ID Card Number Recognition in Android Financial APP, Lecture Notes in Computer Science. 2017. V. 10135. P. 205–213.
8. *Liu L., Huang L., Xue J.* Identification of serial number on bank card using recurrent neural network, Ninth International Conference on Graphic and Image Processing (ICGIP), Qingdao, China 14–16 October 2017. P. 319–325.
9. *Арлазаров В.В., Булатов К.Б., Карпенко С.М.* Метод определения надежности распознавания тисненых символов. Труды Института системного анализа РАН. 2013. Т. 63. № 3. С. 117–122.
10. *Puybareau E., Geraud T.* Real-Time Document Detection in Smartphone Videos. Proceedings of the 24th IEEE International Conference on Image Processing (ICIP), France, October 2018. P. 1498–1502.
11. *Viola P., Jones M.* Rapid object detection using a boosted cascade of simple features. Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Kauai, HI, USA, USA 8–14 December 2001, Section 1. P. 511–519.

12. *Sermanet P., Eigen D., Zhang X., Mathieu M., Fergus R., LeCun Y.* OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. International Conference on Learning Representations (ICLR), Banff, Canada 14–16 April 2014. P. 1055–1061.

13. Copyright 2011–2014, *OpenCV dev team*, 2019. OpenCV документация / Фильтрация изображений. https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#getgaussiankernel

14. *Mason S.J., Graham N.E.* Areas beneath the relative operating characteristics (ROC) and relative operating levels (ROL) curves: Statistical significance and interpretation. / S.J. Mason, N.E. Graham // Q. J. R. Meteorol. Soc. 2002. V. 128. P. 2145–2166.

15. CardIO official site. Available: https://www.card.io

ПРОГРАММИРОВАНИЕ, 2020, № 6, с. 16–29

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.424.64

ПОДХОД К ОБРАБОТКЕ ПУСТЫХ УЗЛОВ ПРИ ПОРЦИОННОЙ ВИЗУАЛИЗАЦИИ ДАННЫХ НА ПРИМЕРЕ ИНСТРУМЕНТА ONTODIA

© 2020 г. Д. С. Раздьяконов[a],*, А. В. Морозов[a],, Д. С. Павлов[b],***, Д. И. Муромцев[a],****

[a] *Факультет программной инженерии и компьютерных технологий, Университет ИТМО, Санкт-Петербург, Кронверкский пр., д. 49, 197101 Россия*

[b] *Metaphacts East Europe, Санкт-Петербург, Большая Разночинная ул., д. 14, 197110 Россия*

**E-mail: ladone3@mail.ru*

***E-mail: am@metaphacts.com*

****E-mail: dp@metaphacts.com*

*****E-mail: mouromtsev@mail.ifmo.ru*

Поступила в редакцию 05.04.2020 г.
После доработки 16.06.2020 г.
Принята к публикации 09.07.2020 г.

Задача ленивой визуализации онтологических графов имеет ограничения. Особые затруднения вызывает необходимость визуализации структур содержащих пустые узлы. В данной статье мы рассмотрим подход к визуализации подобных структур, реализованный в инструменте Ontodia, рассмотрим ограничения данного подхода, а также альтернативные пути решения.

DOI: 10.31857/S0132347420060060

1. ВВЕДЕНИЕ

Среди исследователей, работающих в области семантических технологий, всем известно, что такое пустые узлы (ПУ), иначе называемые Blank nodes. Они используются для представления объектов, не имеющих постоянных идентификаторов, то есть IRI (Internationalized Resource Identifier – интернационализированный идентификатор ресурса). Примером использования может быть, например, представление класса, являющегося объединением нескольких классов, или представление коллекции. В случае представления объединений, например, конструкции `owl:unionOf`, получившийся класс будет пустым узлом, который ссылается на RDF list – структуру, в которой все промежуточные элементы, включая корневой, являются пустыми узлами, а перечисляемые элементы – обычными узлами с читаемыми идентификаторами (рис. 1). То же самое касается представления конструкции `owl:Restriction` (рис. 2). Подразумевается, что подобный узел не имеет значения без указания, на какое свойство вводится ограничение, а также без ограничения на значение.

Отсутствие идентификаторов у пустых узлов вполне оправдано, однако с технической точки зрения это является помехой, так как отсутствие идентификаторов означает невозможность ссылаться на такие узлы. Получается, что ПУ нельзя воспринимать отдельно от структуры, частью которой они являются, тем не менее в некоторых

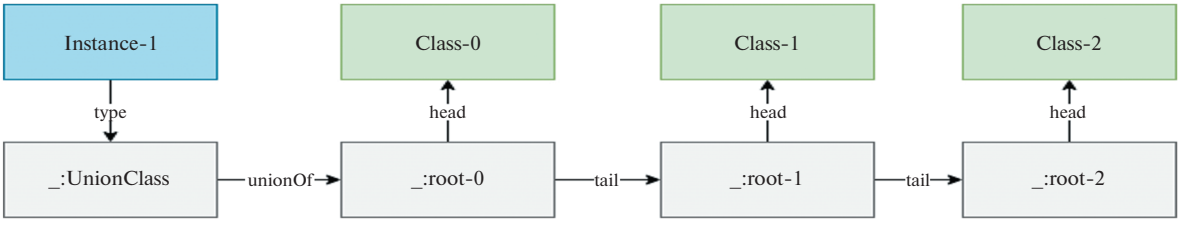


Рис. 1. Представление конструкции RDF list.

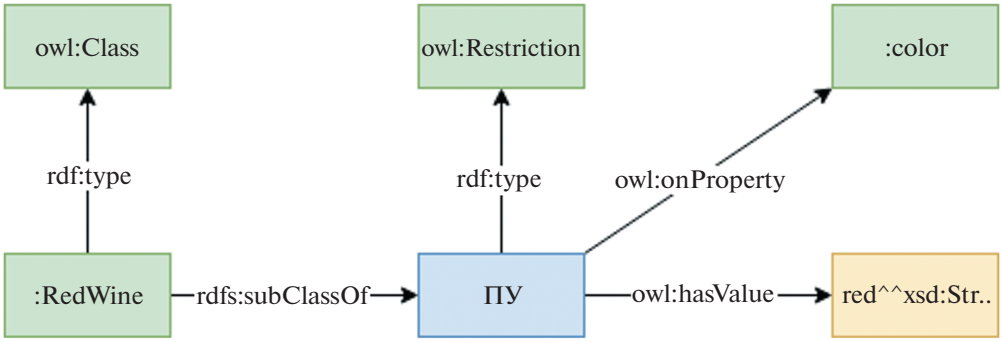


Рис. 2. Представление конструкции `owl:Restriction`.

случаях именно это является сугубо необходимым.

Подобная необходимость, например, возникает, когда перед нами встает задача визуализации графа вкупе с возможностью сохранять и восстанавливать из сохраненного файла или иного артефакта ранее созданную визуализацию. Задача создания инструмента визуализации графов, содержащих ПУ, не является очень трудоемкой до тех пор, пока всю структуру мы получаем одной порцией, но в случае с инструментом Ontodia [1], мы сталкиваемся с ленивой визуализацией, где данные получаются порциями, встает также вопрос о том какие данные уже визуализированы, а какие нет, и здесь отсутствие идентификаторов у пустых узлов играет ключевую роль. При подобной визуализации мы не можем однозначно сказать является ли узел, пришедший вместе с новой порцией данных, тем же самым узлом, который уже отображен на диаграмме, или это иной узел, имеющий тот же набор связей и тот же тип. Подобная проблема возникает, когда мы пытаемся восстановить сохраненную ранее визуализацию, и в этом случае уже не важно, используется ленивая загрузка, или визуализация была построена из одной порции данных.

В данной статье мы рассмотрим, как в инструменте Ontodia решаются вопросы построения визуализации онтологического графа, содержащего ПУ, в условиях ленивой загрузки данных, а также вопрос сохранения и восстановления подобной визуализации (диаграммы).

Визуализация в целом и предложенный подход в частности могут быть полезны в ряде задач по протоке логических выражений, рассмотренных в таких статьях как "Визуализация знаний на основе семантической сети" [2] и "Методы ускорения логического вывода в продукционной модели знаний" [3].

2. СУЩЕСТВУЮЩИЕ ПОДХОДЫ

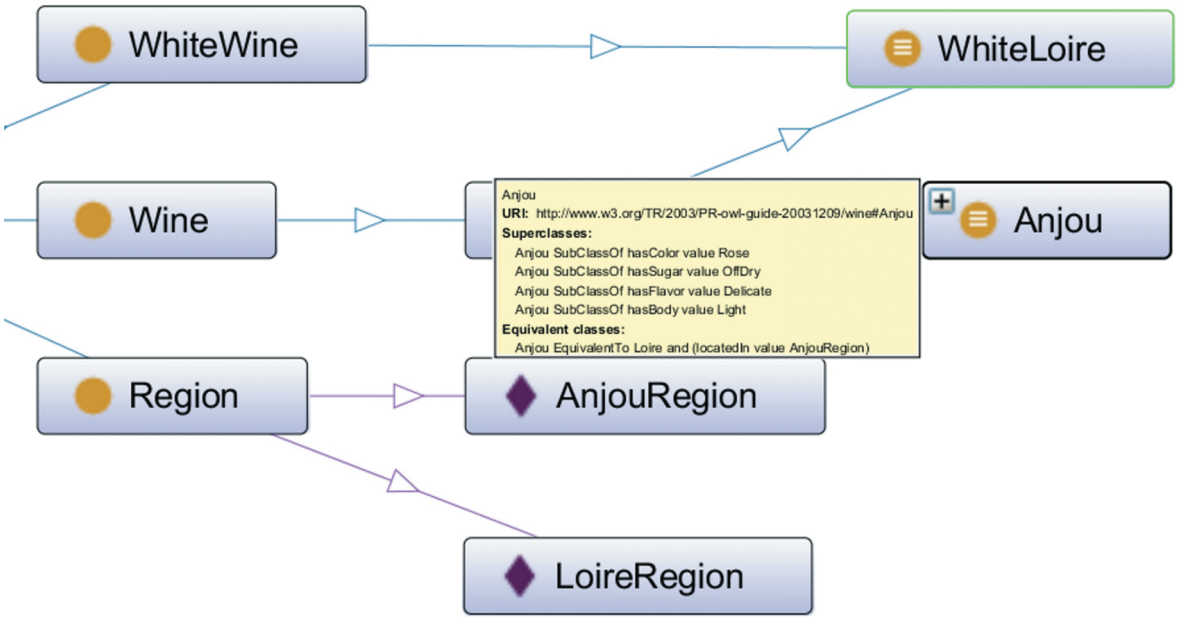
Здесь и далее для примеров используются данные, представленные на листинге 1 на странице 4. Данные являются частью онтологии вин `http://www.w3.org/TR/owl-guide/wine.rdf`. Для наглядности одна из связей продублирована, и ее идентификатор заменен на нестандартный: `owl:intersectionOf` → `<http://example.com/unknownTypeOfProperty>`. Это сделано, чтобы показать, как средства визуализации обрабатывают нестандартные случаи.

Есть несколько подходов к обработке ПУ. Некоторые подходы включают непосредственно решение задачи ленивой визуализации онтологического графа, другие также решают задачу сравнения графов, содержащих ПУ, но преследуют другие цели. Так Ontop Protégé [4] для работы с ПУ вводит ограничение на свойства ПУ – каждый ПУ должен иметь литеральное свойство `rdfs:label`, которое, очевидно, выступает в качестве постоянного идентификатора ПУ. Другой плагин OntoGraph Protégé позволяет визуализировать ПУ, однако нестандартным образом. В OntoGraph ПУ визуализируются в виде аннотаций, как это представлено на рис. 3a. На рис. 3a видно, что аннотации представляются в виде всплывающих окон, при этом конструкция `owl:Restriction` представляется как свойство `Superclasses`, а `owl:IntersectionOf` отображается в виде `Equivalent classes`. Иначе OntoGraph Protégé разбирается с отображением нестандартных структур. На рис. 3b видно, что нестандартная связь к структуре RDF list выделяется в отдельный блок `Annotations`, после чего отображается только первый ПУ структуры.

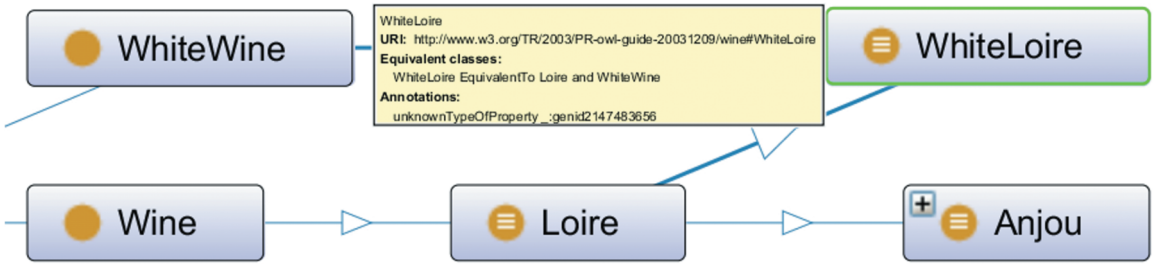
В некоторых инструментах, таких как Graffoo [5], отсутствует поддержка визуализации ПУ, и при рендеринге ПУ исключаются из визуализации. WebVOWL [6] при визуализации онтологий, содержащих ПУ, осуществляет индивидуальный подход

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix wines: <http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#> .

<http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine> a owl:Ontology ;
  rdfs:label "Wine Ontology" .

wines:Anjou a owl:Class ;
  rdfs:subClassOf [ a owl:Restriction ; owl:onProperty wines:hasColor ; owl:hasValue wines:Rose ],
  [ a owl:Restriction ; owl:onProperty wines:hasBody ; owl:hasValue wines:Light ],
  [ a owl:Restriction ; owl:onProperty wines:hasFlavor ; owl:hasValue wines:Delicate ],
  [ a owl:Restriction ; owl:onProperty wines:hasSugar ; owl:hasValue wines:OffDry ] ;
  owl:intersectionOf ( wines:Loire _:blank6 ) .

wines:Loire a owl:Class .
_:blank6 a owl:Restriction ; owl:onProperty wines:locatedIn ; owl:hasValue wines:AnjouRegion .

wines:Loire a owl:Class ; owl:intersectionOf ( wines:Wine _:blank9 ) .
wines:Wine a owl:Class .
wines:LoireRegion a wines:Region ; wines:locatedIn wines:FrenchRegion .

wines:AnjouRegion a wines:Region ; wines:locatedIn wines:LoireRegion .
_:blank6 a owl:Restriction ; owl:onProperty wines:locatedIn ; owl:hasValue wines:AnjouRegion .
_:blank9 a owl:Restriction ; owl:onProperty wines:locatedIn ; owl:hasValue wines:LoireRegion .

wines:WhiteLoire a owl:Class ; owl:intersectionOf ( wines:Loire wines:WhiteWine ) ;
      <http://example.com/unknownTypeOfProperty> ( wines:Loire wines:WhiteWine ) .
wines:WhiteWine a owl:Class .
```

Листинг 1: Представление конструкций owl:IntersectionOf и owl:Restriction в формате Turtle



(a) `owl:IntersectionOf` и `owl:Restriction`



(b) Нестандартный случай, конструкция RDF list

Рис. 3. Представление конструкций в инструменте OntoGraph Protégé.

к визуализации известных структур, однако список поддерживаемых структур ограничен. Так, например, конструкция `owl:intersectionOf` представляется в виде топологии звезда, где центральный узел является пересечением других классов топологии (см. рис. 4), при этом WebVOWL практически игнорирует конструкции `owl:Restriction`, отображая только связи присоединенные к базовой сущность `owl:Thing`. Нестандартные случаи игнорируются. Стоит также отметить, что WebVOWL сосредотачивается в большей степени на визуализации классов, нежели экземпляров классов.

Подобным путем индивидуального подхода идет инструмент OWLGrEd [7]. Конструкция `owl:intersectionOf` визуализируется в виде дерева. Конструкция `owl:Restriction` отображается в теле целевого элемента в виде аннотации (рис. 5a). В отличие от WebVOWL, OWLGrEd способен рисовать экземпляры классов, выделяя их зеленым цветом, однако не справляется с отображением нестандартных случаев. Конструкции типа RDF list игнорируются, если не являются частью конструкции `owl:IntersectionOf` (рис. 5b).

С другой стороны к данному вопросу подходит php-библиотека EasyRdf [8].

При визуализации EasyRdf максимально точно передает структуру онтологического графа, но не предпринимает попыток индивидуальной визуализации общеизвестных структур (см. рис. 6a). На рис. 6a представлена только часть результата. Это сделано вследствие того, что EasyRdf не предоставляет возможность изменять результат визуализации, и полный результат будет сложен для восприятия. Весь граф единомоментно помещается в память браузера, что дает возможность ссылаться на ПУ и производить визуализацию целевых структур. Плюсом данного подхода является то, что нестандартные случаи отображаются в точность так, как они определены (см. рис. 6b). Рассматривая примеры OWLGrEd, WebVOWL,

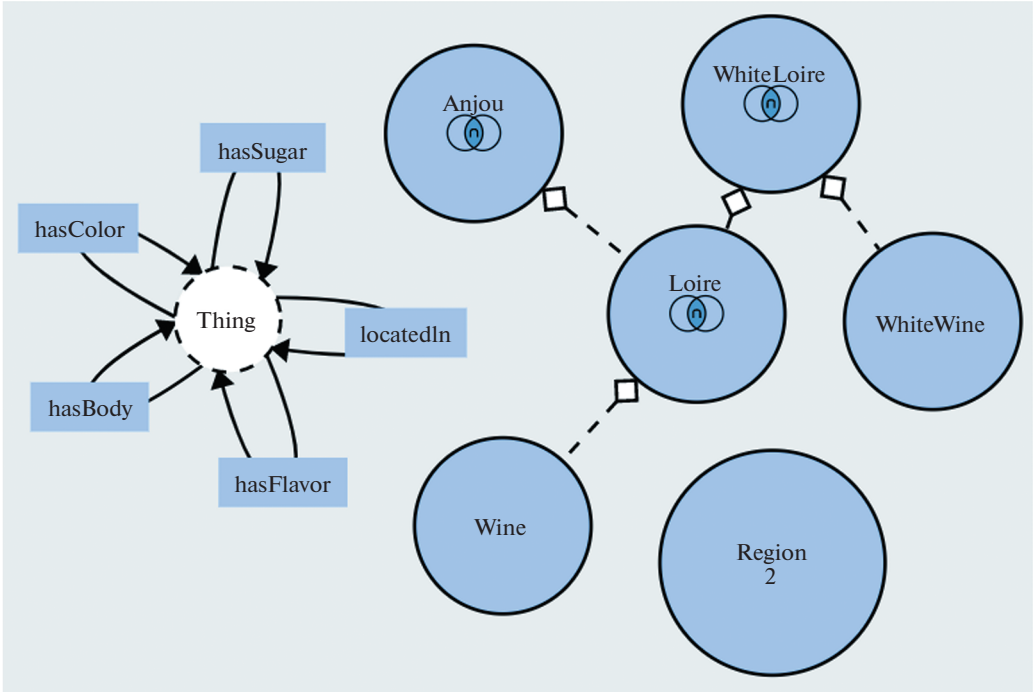


Рис. 4. Конструкции `owl:Restriction` и `owl:IntersectionOf` в инструменте WebVOWL.

следует сказать, что индивидуальный подход имеет свои преимущества при визуализации, если же его совмещать с глобальным подходом, как это делает EasyRdf, то мы получим комфортную визуализацию знакомых структур, совмещенную с возможностью универсальной визуализации.

Данный подход, например, осуществляет инструмент работы с онтологиями TopBraid Composer [9] (см. рис. 7a и 7b). Как можно видеть из результатов визуализации, TopBraid индивидуально подходит к отображению конструкций `owl:IntersectionOf` и `owl:Restriction` (см. рис. 7a), отображая ограничения и пересечения в теле элементов. Подобным образом TopBraid подходит к отображению конструкций RDF list, где элементы списка по порядку отображаются в теле головы списка, но в то же время инструмент способен показать всю структуры целиком, отображая пустые узлы в виде набора RDF list соединенных связями rdf:rest (см. рис. 7b). При этом весь объем данных загружается в память приложения и индексируется в соответствие с задачей, что позволяет порционно визуализировать граф. Вся модель хранится в памяти компьютера пользователя, поэтому визуализация ограничена ресурсами компьютера. Большие базы знаний, такие как Wikidata и DBpedia, не могут быть отображены подобным образом.

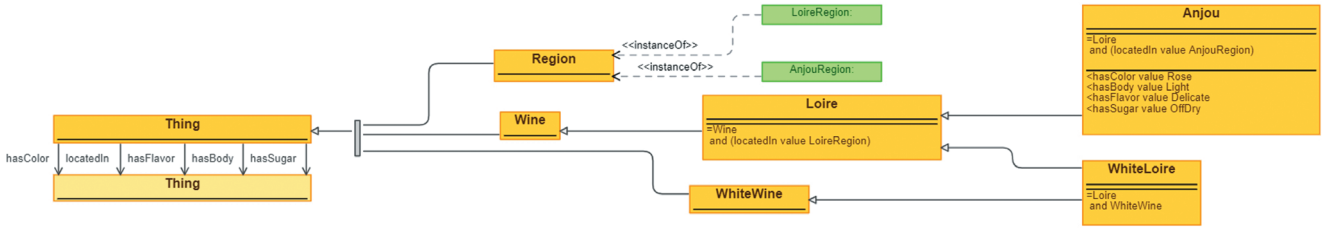
Что же касается инструмента Ontodia, здесь, как и в TopBraid, совершена попытка совместить два подхода, при этом индивидуальный подход в текущей реализации очень беден и включает только обработку конструкций типа RDF list, остальные конструкции визуализируются в рамках глобального подхода (см. рис. 8a). Как можно видеть из рисунка, для всех элементов RDF list проводятся дополнительные связи, указывающие на голову списка, а также указывается порядковый номер элемента в списке. То же самое делается и для хвоста списка, который, в свою очередь, является самостоятельным списком. Такой подход помогает легко справляться с визуализацией нестандартных случаев (см. рис. 8b). Стоит отметить, также, что загрузка данных в инструменте осуществляется лениво, загружается только визуализированная часть онтологического графа, что позволяет работать с большими данными.

Исходя из вышеперечисленного, получаем следующую сводную табл. 1, которая описывает подходы рассмотренных инструментов к визуализации структур, содержащих ПУ.

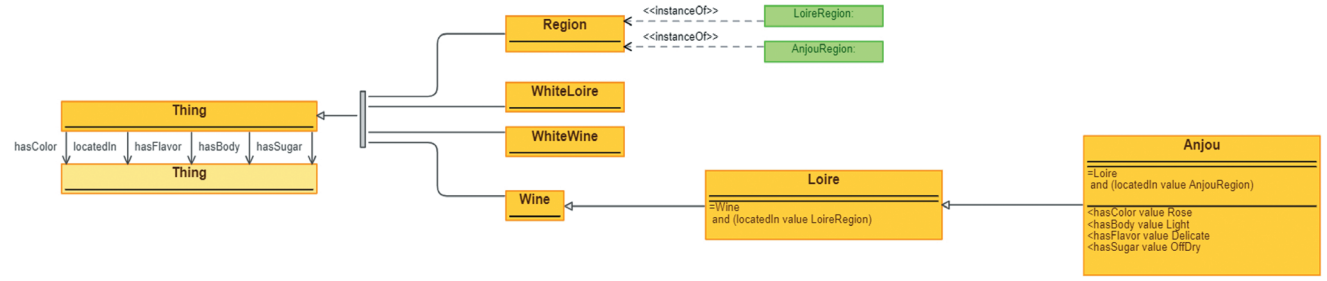
3. ОПИСАНИЕ РЕШЕНИЯ

3.1. Циркуляция данных в ONTODIA

Ленивая загрузка данных графа в инструменте Ontodia осуществляется средствами объекта `Data`



(a) `owl:IntersectionOf` и `owl:Restriction`



(b) Нестандартный случай, конструкция RDF list

Рис. 5. Представление конструкций в инструменте OWLGrEd.

`Provider`. `DataProvider` воплощает идею паттерна проектирования Data Access Object [10]. `DataProvider` отвечает за выбор схемы и данных из базового хранилища, а также за перевод загруженных данных во внутреннюю модель Ontodia. Кроме того, на этапе преобразования данных есть возможность изменять структуру данных в соответствии с потребностями конкретного приложения. Это включает, например, группирование отдельных узлов в таблицы в пользовательском интерфейсе, группирование узлов в суперузлы или свертывание путей между ними. В том числе `DataProvider` позволяет лениво подгружать данные в ответ на запросы.

Возможные запросы данных к `DataProvider` описаны в табл. 2. Большинство запросов используются в специфичных только для Ontodia случаях и существуют для оптимизации циркуляции данных в инструменте для эффективного отображения элементов интерфейса.

Здесь и далее под дополнительной информацией мы будем понимать ту информацию, которая не влияет на топологическую структуру онтологического графа. Иными словами: названия элементов, типы элементов и связей, литеральные свойства элементов.

3.2. Общее описание решения

Решение, которое предлагает Ontodia для отображения ПУ, состоит в том, чтобы ввести стадию предобработки запросов и выдать контекстно зависимые идентификаторы пустым узлам. Стадия предобработки включает в себя несколько шагов: сбор контекста, формирование контекстно зависимых идентификаторов, сохранение результатов предобработки для последующей выдачи. Идентификаторы должны быть составлены так, чтобы было возможно восстановить контекст напрямую из каждого сгенерированного идентификатора и однозначно сравнить узлы.

Контекст определяется следующим образом: *Контекст для целевого пустого узла – это подграф основного графа, который включает целевой ПУ, а также транзитивно все ПУ, заключенные между непустыми узлами, окружающими данный подграф и включающий их* (*см. рис. 9*). То есть контекст для целевого ПУ – это граф, который содержит целевой ПУ и всех его соседей и соседей его соседей, вплоть до первого непустого узла (НПУ).

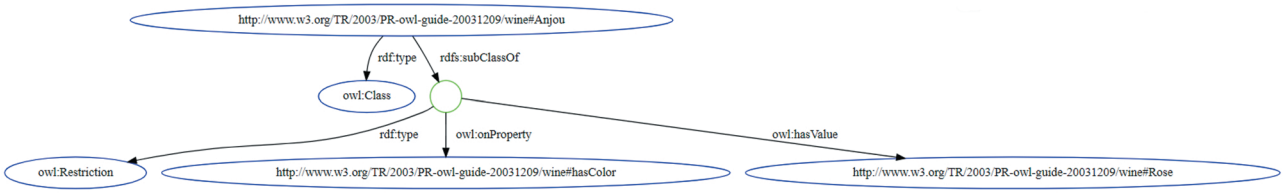
Далее, чтобы подробней описать метод, ответим на следующие вопросы:

1. В какой момент и где производится предобработка данных?

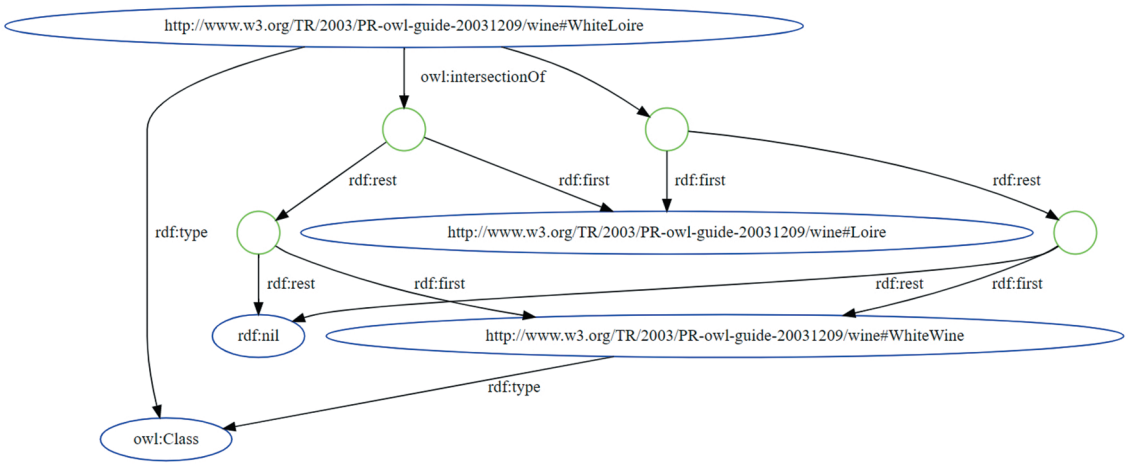
2. Как собирается контекст?

3. Как создаются контекстно зависимый идентификатор?

4. Где и когда происходит выдача результатов?



(a) `owl:Restriction`



(b) Нестандартный случай, конструкция RDF list и `owl:IntersectionOf`

Рис. 6. Представление конструкций в инструменте EasyRdf.

3.3. Где производится предобработка информации

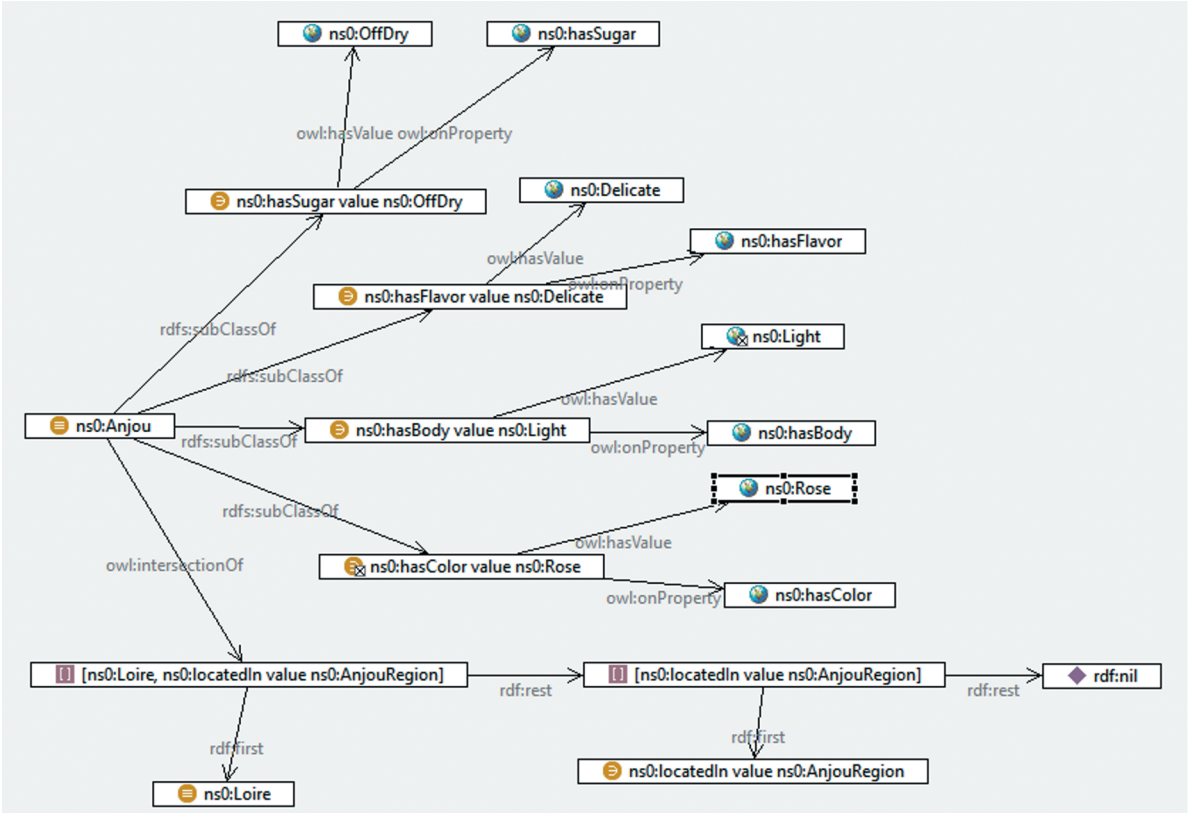
Весь обмен данными между `DataProvider` и диаграммой сводится к запросам диаграммой (у `DataProvider`) связей и элементов, где элементы – это классы и их экземпляры, а также дополнительной информации об элементах и связях. Под запросами на дополнительную информацию понимаются следующие запросы:

- запрос всех возможных типов связей в графе без привязки к конкретным узлам;
- запрос свойств классов и элементов, иначе называемых `DataProperty`;
- запрос свойств связей.

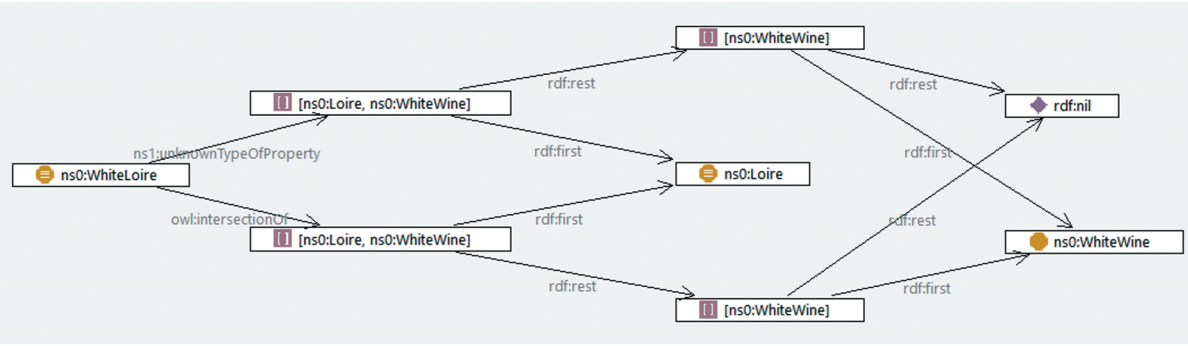
Для выполнения поставленной нами задачи сосредоточимся на тех запросах, которые оперируют узлами, потому что дополнительная информация никак не влияет на структуру онтологического

Таблица 1. Инструменты визуализации онтологических графов

Инструмент	Инд. виз.	Глоб. виз.	Порционно	Загрузка
OntoGraph Protege	да	нет	да	Целиком
Graffoo	нет	нет	да	Целиком
WebVOWL	да	нет	нет	Целиком
OWLGrEd	да	нет	нет	Целиком
EasyRdf	нет	да	нет	Целиком
TopBraid Composer	да	да	да	Целиком
Ontodia	да	да	да	Лениво



(a) `owl:IntersectionOf` и `owl:Restriction`



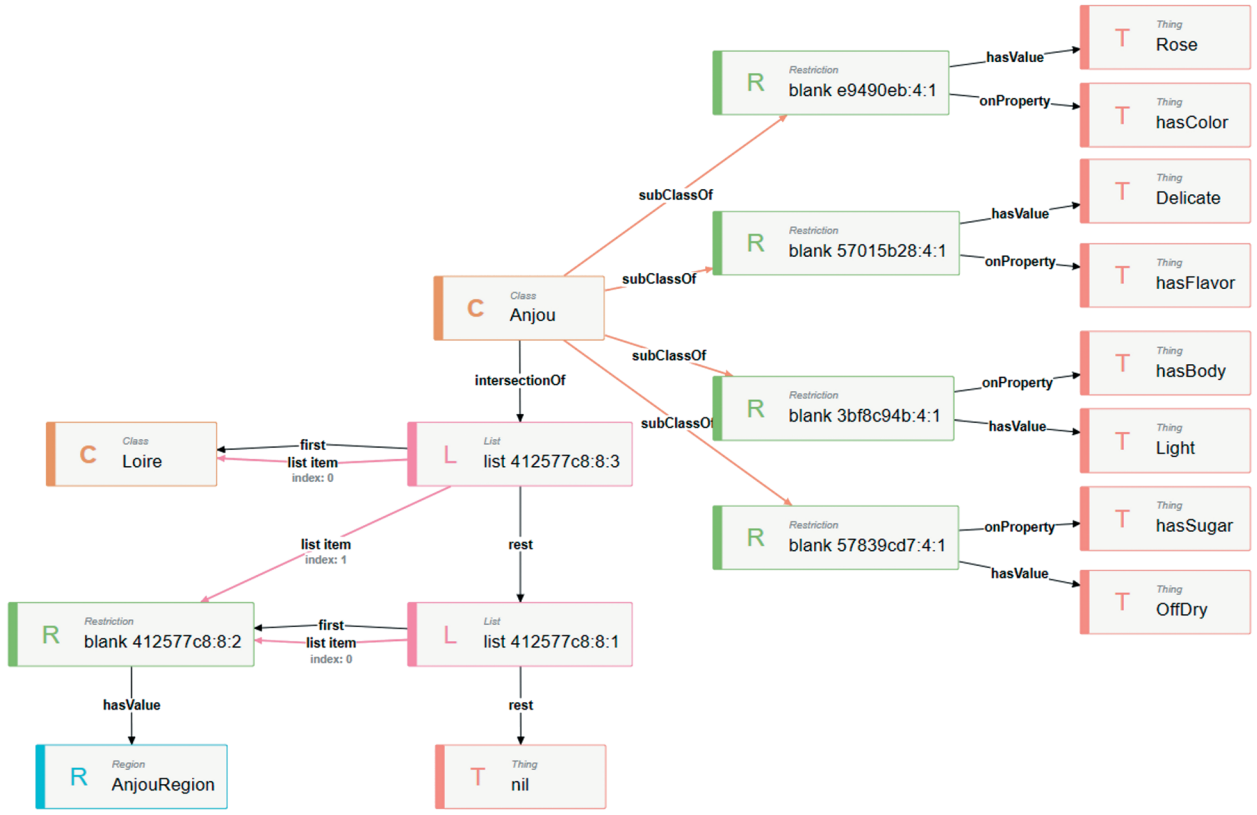
(b) Нестандартный случай, конструкция RDF list и `owl:IntersectionOf`

Рис. 7. Представление конструкций в инструменте TopBraid Composer.

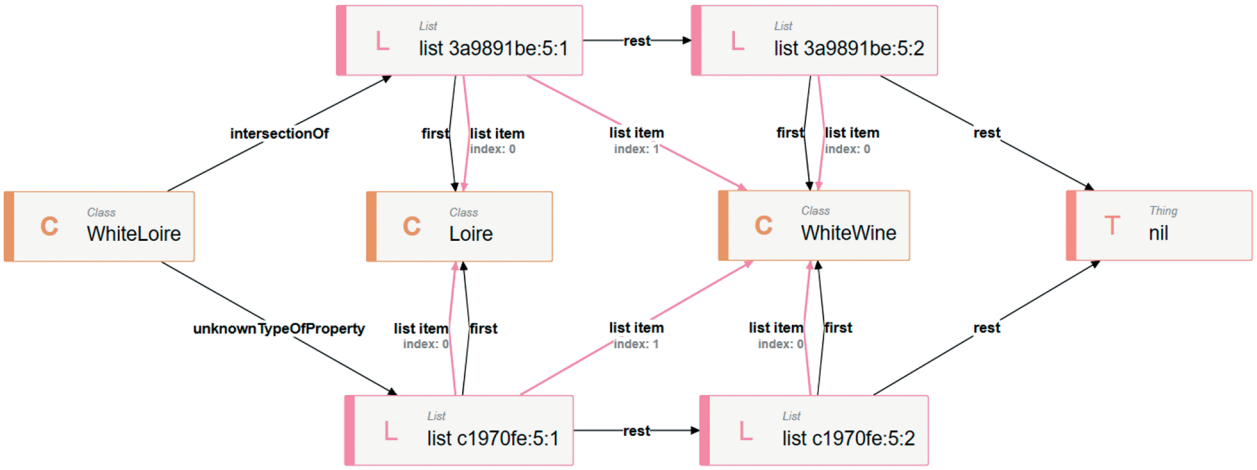
графа, а связи, в случае ПУ, будут закодированы вместе с контекстом в IRI. Из списка запросов, которые используются Ontodia (см. раздел “Циркуляция данных в Ontodia”) для получения данных, мы можем выделить те, которые оперируют исключительно вершинами графа:

- classTree – Запрос дерева классов/типов;
- filter – Запрос списка узлов с указанием условий поиска.

Остальные запросы или возвращают информацию о связях, или оперируют дополнительной информацией. В ходе разработки также было принято решение о том, что дерево классов не должно включать ПУ, так как ПУ обычно не имеют читаемых идентификаторов и не имеют смысла вне структуры, частью которой являются. Соответственно, единственной точкой, в которой мы можем производить предобработку данных, является filter-запрос. Этот запрос используется в Ontodia везде, где нужно получить список элементов – меняются лишь параметры фильтрации. Предобработка информации происходит сразу после получения результатов запроса. Ре-



(a) `owl:IntersectionOf` и `owl:Restriction`



(b) Нестандартный случай, конструкция RDF list и `owl:IntersectionOf`

Рис. 8. Представление конструкций в инструменте Ontodia.

зультаты представляют собой массив троек [subject, predicate, object]. Прежде чем вернуть результаты, в запросе выделяются тройки, содержащие в качестве объекта или субъекта ПУ, они обрабатываются отдельно – это детально разобрано в следующем разделе.

3.4. Получение контекста

Сбор контекста осуществляется путем рекурсивного формирования SPARQL-запроса всего контекста и последующего его выполнения. Допустим, используя функцию `filter DataProvider`, мы получили ПУ. Результат поступает на

Таблица 2. Возможные запросы данных к DataProvider

`classTree`	Запрос дерева классов/типов
`linkTypes`	Запрос возможных типов связей в графе
`classInfo`	Запрос дополнительной информации о классе, если имеется
`linkTypesInfo`	Запрос данных о связи
`elementInfo`	Запрос данных об элементах
`linksInfo`	Запрос связей между элементами
`linkTypesOf`	Запрос списка типов входящих/исходящих связей элемента
`filter`	Запрос списка элементов с указанием условий поиска

конвейер предобработки. Здесь нужно пояснить, что функция `filter` в большинстве случаев осуществляет поиск с привязкой к целевому элементу, то есть поиск соседей целевого элемента или фильтрацию списка возможных связей для целевого элемента. Исключением является только поиск по ключевому слову, который может осуществляться без указания целевого элемента. Пустой узел не может быть найден с использованием поиска по ключевому слову, поэтому, когда ПУ поступает на конвейер, вместе с ним поступает IRI элемента, для которого выполнялся поиск, и тип связи между ПУ и целевым элементом. На основе этих данных создается первичный SPARQL-запрос. Далее запрос выполняется, и его результаты проверяются на наличие ПУ. Если новые ПУ не обнаружены, то контекст считается завершенным, в противном случае результат используется для расширения первичного запроса, после чего осуществляется выполнение расширенного SPARQL-запроса и первый шаг повторяется. Цикл происходит до тех пор, пока мы не получим результат без ПУ, или контекст не будет включать весь граф. Результат последнего запроса и будет искомым контекстом.

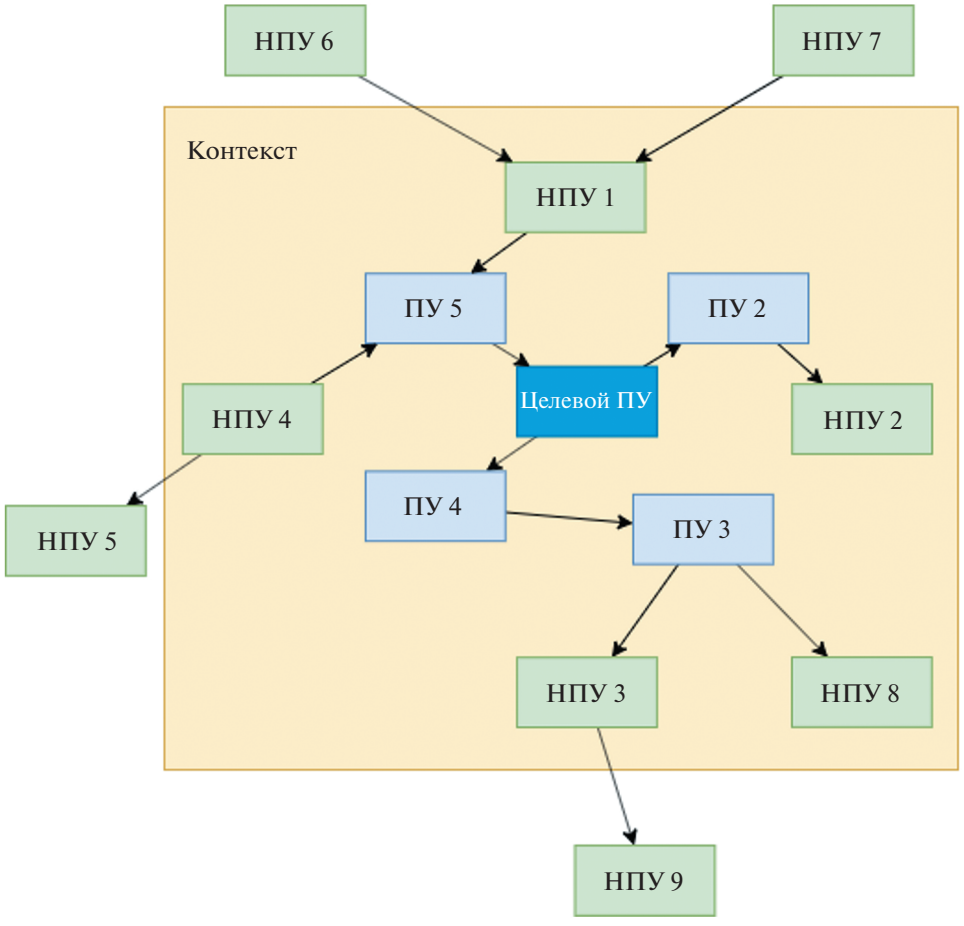


Рис. 9. Представление контекста ПУ.

3.5. Контекстно зависимые идентификаторы

Когда контекст собран, начинается фаза формирования контекстно зависимых идентификаторов. Идея состоит в том, чтобы создать идентификатор, из которого в дальнейшем можно будет вычислить контекст. Для этого в Ontodia контекст преобразовывается в хэш и используется в качестве идентификатора. При этом хэш-функция реализует следующие шаги:

1. Граф контекста преобразуется в каноничную форму (см. Листинг 2). Метод преобразования RDF-графа в каноничную форму описан в статье “Canonical Forms for Isomorphic and Equivalent RDF Graphs: Algorithms for Leaning and Labelling Blank Nodes” [11].

2. IRI элементы кодируются JavaScript-функцией `encodeURIComponent`, чтобы данный хэш можно было использовать как часть IRI у компонента.

3. После чего из графа извлекается словарь терминов и выделяется специальный массив, описывающий типы каждого термина (N – NamedNode, B – BlankNode, L – Literal (Строка с указанием языка), D – Literal (Строка с указанием типа), V – Variable, G – DefaultGraph).

4. Словарь и граф, представленный в виде набора четверок (quads), сжимаются и кодируются с

```
{
    "quads": [{
            "subject": {"value": "b3"},
            "predicate": {"value": "http://www.w3.
              org/1999/02/22-rdf-syntax-ns#first"},
            "object": {"value": "http://www.w3.org/
              TR/2003/PR-owl-guide-20031209/wine#
              Loire"},
            "graph": {"value": ""}
        },
        ...
        {
            "subject": {"value": "b4"},
            "predicate": {"value": "http://www.w3.
              org/1999/02/22-rdf-syntax-ns#type"},
            "object": {"value": "http://www.w3.org/2
              002/07/owl#Class"},
            "graph": {"value": ""}
        }, {
            "subject": {"value": "b4"},
            "predicate": {"value": "http://www.w3.
              org/2002/07/owl#intersectionOf"},
            "object": {"value": "b3"},
            "graph": {"value": ""}
        },
        ...
    ], "pointer": {  "value": "b4" }
}
```

Листинг 2: Часть канонизированного графа контекста в каноничной форме, включающая указатель на целевой элемент

```
[[
    [0,"http%3A%2F%2Fwww.w3.org%2F",[0,"1999%2F02%2F
       22-rdf-syntax-ns%23",[0,"first",1,"nil",1,"
       rest",1,"type",1],"2002%2F07%2Fowl%23",[0,"
       Class",1,"Restriction",1,"equivalentClass",1,
       "hasValue",1,"intersectionOf",1,"onProperty",
       1],"TR%2F2003%2FPR-owl-guide-20031209%2Fwine%
       23",[0,"Anjou",[1,"Region",1],"Loire",1,"
       locatedIn",1]]],
    ["N",0,"N",1,"N",2,"N",3,"N",4,"N",5,"N",6,"N",7
       ,"N",8,"N",9,"N",10,"N",11,"N",12,"N",13]
],
[-1,0,-2,-1,2,1,-3,0,12,-3,2,-1,-4,3,4,-4,8,-3,-2,3,
   5,-2,7,11,-2,9,13,10,6,-4]]
```

Листинг 3: Компактное представление графа контекста в формате JSON, полученное из IRI элемента

```
ontodia:blank:sparql2:4:(((0:'http%3A%2F%2Fwww.w3.
   org%2F':(0:'1999%2F02%2F22-rdf-syntax-ns%23':(0:'
   first':1:'nil':1:'rest':1:'type':1):'2002%2F07%2
   Fowl%23':(0:'Class':1:'Restriction':1:'
   equivalentClass':1:'hasValue':1:'intersectionOf':
   1:'onProperty':1):'TR%2F2003%2FPR-owl-guide-20031
   209%2Fwine%23':(0:'Anjou':(1:'Region':1):'Loire':
   1:'locatedIn':1))):('N':0:'N':1:'N':2:'N':3:'N':4
   :'N':5:'N':6:'N':7:'N':8:'N':9:'N':10:'N':11:'N':
   12:'N':13)):(-1:0:-2:-1:2:1:-3:0:12:-3:2:-1:-4:3:
   4:-4:8:-3:-2:3:5:-2:7:11:-2:9:13:10:6:-4))
```

Листинг 4: Пример сгенерированного IRI

использованием префиксного дерева [12] (см. Листинг 3).

5. Полученные элементы объединяются в массив.

6. В полученной строке заменяются символы: [→ (,] →), , → : , " →".

7. Далее, для получение финального IRI (см. Листинг 4) к полученному хэшу добавляется индивидуальный индекс и префикс. Префикс различен для разных типов узлов:

(a) “ontodia:blank:” – если одиночный ПУ;

(b) “ontodia:list:List” – если RDF list.

Последнее делается для того, чтобы по IRI элемента можно было однозначно определить является ли IRI закодированным контекстом или нет, а также для возможности последующего индивидуального отображения списков.

Чтобы получить контекст из закодированного IRI, достаточно выполнить описанные шаги в обратном порядке.

3.6. Выдача результатов предобработки

После того как идентификаторы сформированы, измененный контекст кладется в локальное

хранилище, которое, также как и основное, реализует интерфейс `DataProvider`. Впервые выдача результатов, очевидно, происходит в функции `filter`, сразу после предобработки, однако это не единственное место. Например, вызов функции `filter` возвращает набор пустых узлов, которые мы добавляем на граф, при этом информация о связях, входящих в контекст, остается неиспользованной. На следующем шаге процесса отрисовки графа, когда выполняется запрос на получение связей между элементами, эта информация становится полезной. Здесь данные о связях уже берутся не из основного хранилища, а из локального. Доступ к локальном хранилищу более эффективен, и данные здесь уже канонизированы. Полный список запросов, в которых используются предварительно обработанные данные, таков:

- `elementInfo`
- `linksInfo`
- `linkTypesOf`
- `filter`

Когда мы загружаем сохраненную диаграмму, происходит обратное. Как только мы встречаем элемент со специальным префиксом "`ontodia:blank:`" или "`ontodia:list:List`", мы выполняем восстановление контекста из IRI и кладем результат в локальное хранилище, после чего возвращаем результат, как это было описано выше.

4. ЗАКЛЮЧЕНИЕ

Разработанный метод решает задачи визуализации и восстановления сохраненного графа. При сохранении графа сохраняются и идентификаторы узлов, а поскольку проблемная часть (контекст с ПУ) кодируется в идентификаторах, она также легко восстанавливается при загрузке сохраненного графа. Метод описан в общем виде и подходит для ленивой визуализации ПУ онтологических графов не только с использованием инструмента Ontodia, но и при работе с другими инструментами визуализации. Следующим шагом развития данного метода может быть формализация обработки специальных структур, использующих ПУ в качестве структурных компонентов:

- Список (RDF list);
- Аксиома (`owl:Axiom`);
- Ограничение (`owl:Restriction`);
- Взаимное различие (`owl:AllDifferent`);
- Объединение (`owl:unionOf`);
- Пересечение (`owl:intersectionOf`);
- Дополнение (`owl:complementOf`);
- Перечисление (`owl:oneOf`) [13].

Например, RDF list может быть предварительно обработан и визуализирован на диаграмме в виде узла-таблицы с сохранением оригинального порядка следования элементов.

В то же время визуализация данных часто граничит с задачей визуального редактирования данных. В этом направлении имеется также ряд трудностей. Например, представленный алгоритм рассматривает онтологию как нечто завершенное и не подлежащее изменению. В случае, если онтологический граф в области контекста ПУ изменится, все идентификаторы узлов потеряют свое значение, и контекст нужно будет собирать повторно. При этом не всегда просто сказать, была ли изменена онтология в области контекста ПУ, для этого нужно сравнить старый граф контекста с новым, что само по себе часто является нетривиальной задачей. В связи с этим интересно будет адаптировать представленный метод для задачи редактирования онтологических графов, содержащих ПУ.

СПИСОК ЛИТЕРАТУРЫ

1. *Mouromtsev D., Pavlov D., Emelyanov Y., Morozov A., Razdyakonov D., Galkin M.* The simple web-based tool for visualization and sharing of semantic data and ontologies. In International Semantic Web Conference (Posters & Demos) (2015).
2. *Бессмертный И.А.* Визуализация знаний на основе семантической сети // Программирование. 2010. № 4. С. 16–24.
3. *Катериненко Р.С., Бессмертный И.А.* Метод ускорения логического вывода в продукционной модели знаний // Программирование. 2011. № 3. С. 76–80.
4. *Gennari J.H., Musen M.A., Fergerson R.W., Grosso W.E., Crubzy M., Eriksson H., Noy N.F., Tu S.W.* The evolution of protg: an environment for knowledge-based systems development // International Journal of Human-Computer Studies. 2003. V. 58. № 1. P. 89–123.
5. *Falco R., Gangemi A., Peroni S., Shotton D., Vitali F.* Modelling owl ontologies with graffoo. In The Semantic Web: ESWC 2014 Satellite Events (Cham, 2014), V. Presutti, E. Blomqvist, R. Troncy, H. Sack, I. Papadakis, and A. Tordai, Eds., Springer International Publishing, 2014. P. 320–325.
6. *Lohmann S., Link V., Marbach E., Negru S.* Webvowl: Web-based visualization of ontologies. P. 154–158.
7. *Cerns K., Ovinnikova J., Liepin R., Grasmanis M.* Extensible visualizations of ontologies in owlgred. In The Semantic Web: ESWC 2019 Satellite Events (Cham, 2019), P. Hitzler, S. Kirrane, O. Hartig, V. de Boer, M.-E. Vidal, M. Maleshkova, S. Schlobach, K. Hammar, N. Lasier-

ra, S. StadtmBEuller, K. Hose, R. Verborgh, Eds., Springer International Publishing. P. 191–196.

8. *Humfrey N.* Easyrdf a php library designed to make it easy to consume and produce rdf. http://www.easyrdf.org.

9. Topbraid composer maestro edition is a modeling tool and an ide for enterprise solutions. learn how to to model ontologies, connect data sources, design queries, and to develop applications that work with semantic models. https://www.topquadrant. com/knowledge-assets/faq/tbc/.

10. *Nock C.* Data access patterns: database interactions in object-oriented applications. Addison-Wesley Boston, 2004.

11. *Hogan A.* Canonical forms for isomorphic and equivalent rdf graphs: Algorithms for leaning and labelling blank nodes. ACM Trans. Web 11, 4 (July 2017).

12. *Гудкова Т.С.* Префиксное сжатие индексов. Информатика: проблемы, методология, технологии. 2019. С. 1310–1314.

13. *Заикин И.А.* Алгоритм сравнения вариантов онтологий. Электронные средства и системы управления. 2010. № 1. С. 132–135.

ПРОГРАММИРОВАНИЕ, 2020, № 6, с. 30–40

ПРОГРАММНАЯ ИНЖЕНЕРИЯ, ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ ПРОГРАММ

УДК 004.42

АЛГОРИТМ АДАПТИВНОГО ИЗМЕНЕНИЯ МЕНЮ ПРОГРАММНОГО ПРИЛОЖЕНИЯ

© 2020 г. Е. Н. Чуйкова[a],*, А. Р. Айдинян[a],, О. Л. Цветкова[a],*****

[a] *Донской государственный технический университет, 344002 Ростов-на-Дону, пл. Гагарина, 1, Россия*

**E-mail: elenchu@mail.ru*

***E-mail: andstyle@mail.ru*

****E-mail: olga_cvetkova@mail.ru*

Поступила в редакцию 24.02.2020 г.
После доработки 17.04.2020 г.
Принята к публикации 01.06.2020 г.

Меню программного приложения представляет собой важную часть его интерфейса для взаимодействия с пользователем. Удовлетворенность пользователя и эффективность использования программного приложения зависит от того, насколько правильно построено меню. Спроектированные разработчиком меню ориентированы на среднестатистического пользователя и не учитывают потребности и особенности отдельных пользователей. Решением данной проблемы является построение адаптивных пользовательских интерфейсов, обладающих способностью подстройки под нужды конкретного пользователя. Предложенный в статье подход к адаптации пользовательского меню программного приложения выполняется на основе модели пользователя, формируемой в результате наблюдения за действиями пользователя в процессе работы с программой. Построены математическая модель меню программного приложения и модель пользователя программного приложения, предложен алгоритм адаптивного изменения меню программного приложения. Благодаря предложенному алгоритму создания ссылок на пункты меню на более высоком уровне и скрытия неиспользуемых пунктов меню удалось уменьшить время активации пунктов меню, что повышает эффективность работы пользователей программного приложения.

DOI: 10.31857/S0132347420060035

1. ВВЕДЕНИЕ

Программный интерфейс, приспособленный к потребностям конкретных пользователей, является важным средством обеспечения эффективности применения программного приложения [1]. Поэтому усилия многих разработчиков направлены на решение данной задачи. В [2] представлена реализация концепции поддержки пользователей в контекстно-зависимом мобильном устройстве с адаптивным пользовательским интерфейсом. В [3] описан подход на основе шаблонов проектирования к реализации адаптивных пользовательских интерфейсов для пользователей с особыми потребностями. В [4] представлена система автоматической адаптации пользовательских интерфейсов, использующая модель идентификации анонимных пользователей программных продуктов и динамический идентификатор для автоматической адаптации интерфейса к потребностям идентифицированного пользователя. В [5] предложена архитектура адаптивной интерактивной системы для приложений из области электронного бизнеса, способной оценивать текущую ситуацию и реагировать на изменение контекста, среды и эмоций пользователя. Система содержит компонент, осуществляющий наблюдение за поведением пользователя в процессе его взаимодействия с системой. Необходимые модификации программного обеспечения выполняются в режиме реального времени. В [6] для решения задачи построения адаптивного пользовательского интерфейса предложена система анализа взаимодействий, которая с помощью вероятностных методов прогнозирует взаимодействия с пользователем, распознает действия пользователя и определяет

его предпочтения на разных уровнях абстракции. В [7] представлен анализ текущих достижений в области адаптивных пользовательских интерфейсов, определены перспективные направления в этой области, предложена таксономия для сравнения различных систем адаптивных пользовательских интерфейсов, выявлены общие принципы их эффективного проектирования, такие как персонализация методов взаимодействия с пользователей на основе учета его предпочтений, фильтрация нерелевантной информации с целью снижения когнитивной перегрузки пользователя, информационное сопровождение пользователя в процессе освоения нового приложения, включая обнаружение и исправление неправильных действий, объяснение новых функций и предоставление информации для упрощения решения задач.

Неотъемлемой составной частью любого пользовательского интерфейса является меню, поэтому построение адаптивного пользовательского интерфейса невозможно без обеспечения возможности оперативной перестройки его меню. Меню современного программного приложения отличается сложной древовидной структурой, в которой пункты меню находятся на разных уровнях. Чем ниже в иерархии расположен пункт меню, тем больше времени требуется на доступ к нему. Также на время доступа к элементу меню влияет количество пунктов в подменю, их последовательность, запоминаемость меню, опыт пользователя [8]. Усилия многих разработчиков были направлены на создание дружественного меню, интуитивно понятного любому пользователю [9]. Однако такое меню ориентировано на среднестатистического пользователя и не учитывает потребности отдельных пользователей. Решением данной проблемы является построение пользовательских меню, обладающих способностью подстройки под нужды конкретного пользователя [10, 11].

В [12, 13] определены четыре вида меню по способу адаптации:

– адаптируемые пользователем;

– адаптируемые пользователем с поддержкой системы;

– адаптивные – весь процесс адаптации управляется системой;

– адаптивные с контролем пользователя – система выполняет адаптацию под наблюдением пользователя.

Таким образом, выделяют адаптивные и адаптируемые меню, отличающиеся тем, что адаптируемые меню пользователь может сам приспосабливать к своим предпочтениям (например, путем выбора параметров, определяющих внешний вид пользовательского интерфейса).

При использовании адаптируемых меню в программном приложении необходимо предусмотреть соответствующие инструментальные средства, позволяющие пользователю изменять меню, добавлять панели инструментов, назначать действия кнопкам панелей инструментов и т.п. [14, 15]. Такие изменения в меню способен внести опытный пользователь, хорошо знающий систему и средства ее модификации.

При автоматической настройке меню программного приложения модифицируется без участия пользователя на основании имеющейся о нем информации (модели пользователя). При этом возможны негативная реакция пользователя и ошибки в предсказании действий пользователя. Поэтому адаптационные изменения должны проводиться с учетом этих факторов [5, 16–18].

Меню определяет набор возможных действий пользователя с объектами программного приложения. Адаптивность меню выражается в изменении этого набора в зависимости от потребностей определенного пользователя. Например, если имеются редко используемые пользователем пункты меню, то меню изменяется путем сокрытия этих элементов.

В ряде работ представлены результаты сравнения производительности адаптивного, адаптируемого и статического меню, приводящие к неоднозначным выводам. Так, в [19] показано, что большинство пользователей отдают предпочтение статическим меню, а в работе [20] утверждается, что адаптируемые меню являются более предпочтительными по сравнению с адаптивными меню, которые, в свою очередь, не хуже статических. Наиболее эффективным является сочетание автоматически адаптируемых меню с элементами ручной настройки [15, 21]. Причем ручная настройка должна выполняться удобным для пользователя образом без входа в специальные режимы настройки меню. Для ускорения доступа к элементам меню выполняется перенесение часто используемых пунктов в верхнюю часть (разделенные меню), выделение часто используемых пунктов (шрифтом, цветом) без перенесения из привычного места [8].

В [22] предложен метод оптимизации иерархических меню, позволяющий определить наиболее подходящее распределение элементов в древовидной структуре меню. Оптимизация выполняется на основе эвристических алгоритмов (имитация отжига и генетические алгоритмы). Однако использование меню с изменяющейся структурой

может снизить производительность работы пользователя вследствие нестабильности меню и дополнительных затрат времени, требуемых пользователю на изучение незнакомого меню [22], поэтому более целесообразно предусмотреть средства ускорения доступа к часто используемым элементам меню без существенного изменения его структуры посредством перераспределения и переупорядочивания элементов меню. Эта идея реализована в предложенном в данной статье алгоритме, обеспечивающем динамическую адаптацию меню посредством создания ссылок на часто используемые пункты меню и добавления этих ссылок в меню вышележащего уровня. По мере активации ссылок создаются ссылки на еще более высоком уровне. Кроме осуществляемой таким образом автоматической адаптации меню предусмотрены средства ручной настройки, предоставляемые пользователю.

2. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ МЕНЮ ПРОГРАММНОГО ПРИЛОЖЕНИЯ

Меню программного приложения характеризуется навигационной структурой, представляющей собой иерархию пунктов меню. Все пункты меню можно разделить на две группы:

1) пункты меню, активирующие команду;

2) пункты меню, содержащие подменю.

Иерархию элементов меню можно представить в виде N-арного дерева – связного ациклического графа. Корень дерева (root) соответствует меню программы. По определению каждый узел дерева, кроме корня, может иметь только одного предка. Узел дерева может порождать несколько потомков, которыми являются узлы, соответствующие элементам меню нижнего уровня. При представлении меню в виде дерева листьями являются элементы множества $R = \{r\}$ – множество пунктов меню, активирующих команду, а узлами, имеющими потомков, – элементы множества $S = \{s\}$ – множество пунктов меню, содержащие вложенные подменю.

Каждый узел дерева – лист $r \in R$ дерева или узел $s \in S$, имеющий потомков, – относится к некоторому уровню иерархии навигационной структуры меню – l^r или l^s соответственно.

Пункт подменю, содержащий элемент $r \in R$, можно описать следующим образом:

$$s^r = (s \in S \mid s \prec r \in R,\ l^r - l^s = 1).$$

Эффективность работы пользователя с меню определяется глубиной уровня иерархии, на котором находится требуемый пользователю пункт меню. Поэтому целесообразно создавать ссылки на пункты меню, активированные K раз, на более высоком уровне иерархии. При активации ранее созданной ссылки более K раз создается новая ссылка на более высоком уровне. Таким образом, в модель меню добавляется множество E, содержащее ссылки на пункты меню. Если количество активаций пункта меню или ссылки на нее равно или превышает некоторое заданное количество K, то на один уровень выше пункта меню создается ссылка на этот пункт меню. Аналогичный принцип распространяется на ранее созданные ссылки при их многократной активации. Вновь созданная ссылка выделяется заметной меткой для того, чтобы пользователь обратил на нее внимание и начал ею пользоваться. Метка исчезает после того как пользователь ее активировал указанное количество раз (обычно один или два раза). В случае, если созданная ссылка не является востребованной (количество активаций менее порогового значения K за указанное время T), то она удаляется. При расчете времени существования ссылок учитывается время работы пользователя в программе, а не календарное время.

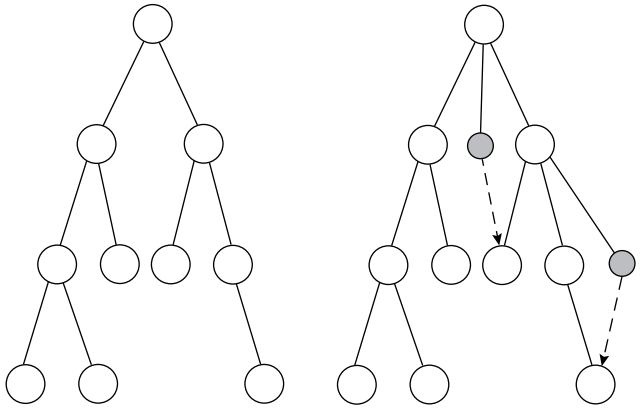


Рис. 1. Структура меню программного приложения в виде дерева до создания ссылок и в виде графа после создания ссылок.

Каждая ссылка $e \in E$ описывается парой $e = (r^e \in R, s^e \in S)$, где r^e – пункт меню, на который указывает ссылка e; s^e – подменю, включающее ссылку e.

Дерево меню с добавленными ссылками фактически превращается в граф, содержащий сокращенные пути к требуемым элементам меню, за счет обхода уровней меню (рис. 1).

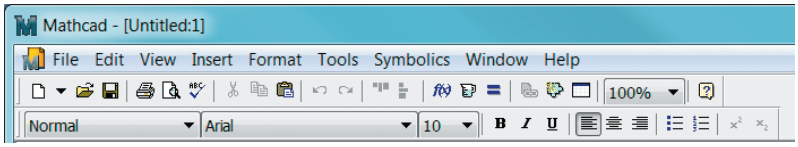


Рис. 2. Пример панелей инструментов в приложении PTC MathCAD.

3. МОДЕЛЬ ПОЛЬЗОВАТЕЛЯ ПРОГРАММНОГО ПРИЛОЖЕНИЯ

Для автоматической перенастройки меню под конкретного пользователя необходима модель пользователя, которая может строиться на основе наблюдений за действиями пользователя и фиксации того, сколько раз к элементам меню обращался пользователь [23, 24].

Пусть $U = \{u\}$ – множество пользователей. Тогда математическую модель отдельного пункта меню с учетом действий пользователя u можно определить следующим образом:

– пункт меню r, активирующий команду, описывается пунктом меню, в котором он содержится, количеством активаций его пользователем u и признаком видимости:

$$r = (s^r, v^r, h^r) \forall r \in R;$$

– ссылка на пункт меню описывается пунктом меню, являющимся объектом ссылки, пунктом меню, содержащим ссылку, количеством активаций ссылки пользователем u, временем ее создания:

$$e = (r^e, s^e, v^e, t^e) \forall e \in E.$$

Здесь $s^r = (s \in S \mid s \prec r, l^r - l^s = 1)$ – подменю, содержащее элемент r; v^r, v^e – число активаций пользователем u элемента меню r и ссылки e соответственно за указанное время работы с программным приложением; h^r – признак видимости элемента r, принимающий одно из двух значений – “виден” (“visible”) или “скрыт” (“hidden”); $r^e = (r \in R \mid e \to r)$ – пункт меню, являющийся объектом ссылки e (запись $e \to r$ означает, что e является ссылкой на элемент r); s^e – подменю, содержащее ссылку e; t^e – время создания ссылки e.

Время обращения к пункту меню зависит от опыта пользователя, от уровня, на котором находится пункт меню, и количества элементов в подменю.

Степень соответствия навигационной структуры меню потребностям пользователя u можно оценить средним временем доступа к элементам r, которое зависит от количества действий пользователя, необходимых для доступа к пункту меню, или от числа уровней навигационной структуры и места требуемого элемента в содержащем его подменю s^r.

Тогда задача перестройки меню сводится к такому изменению его навигационной структуры для пользователя u, при которой достигается минимум значения $\sum_{r \in R} v^r \cdot \tau^r$, где τ^r – среднее время активации пункта меню r. При этом накладываются ограничения на максимальное количество элементов, вынесенных из каждого подменю на один из уровней выше. Оно не должно превышать значения когнитивной константы C, определяющей условия эффективного восприятия пользователем элементов меню и комфортности его работы с программным приложением.

Решение данной задачи позволит получить оптимальную для конкретного пользователя навигационную структуру меню, обеспечивающую повышение комфортности и производительности работы пользователя.

4. АЛГОРИТМ АДАПТИВНОЙ ПОДСТРОЙКИ ПУНКТОВ МЕНЮ

Алгоритм предназначен для адаптивной подстройки меню к конкретному пользователю и должен обеспечить настройку меню с учетом действий пользователя, характеризуемых количеством активаций каждого пункта меню.

Целью предлагаемого алгоритма подстройки меню является сокращение суммарного времени обращения к элементам меню путем вынесения ссылок на часто используемые элементы меню на более высокий уровень иерархии и скрытия редко используемых пунктов меню.

Такой подход для ускорения обращения к командам реализуется с помощью панелей инструментов, обычно расположенных в окне программы под главным меню. Пример приведен на рис. 2.

Панели инструментов позволяют получить доступ к команде за один щелчок мыши, поскольку команды расположены непосредственно в окне формы. Но такое расположение “быстрых кно-

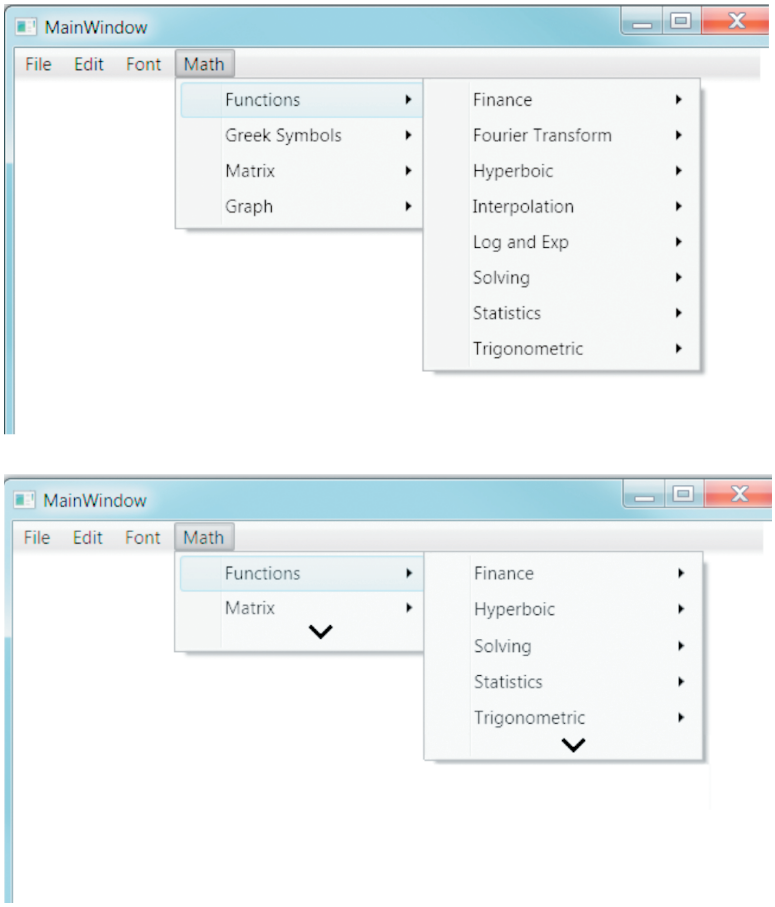


Рис. 3. Внешний вид меню программного приложения без скрытых пунктов меню и с усеченным вариантом меню.

пок” имеет свои недостатки: занимает место в окне, имеются кнопки, которые многим не нужны, требует ручной настройки для подгонки к пользователю, не всегда очевидно соответствие команды изображенной иконке. Даже автоматическое скрытие редко используемых команд в панели инструментов не всегда позволяет преодолеть выше перечисленные недостатки.

Предлагаемый алгоритм решает задачу сокращения времени активации команд путем перестройки меню под конкретного пользователя.

Количество уровней меню и количество пунктов в каждом подменю задается разработчиком изначально и в дальнейшем не изменяется.

Предлагаются два способа сокращения времени доступа:

1. Повышение уровня, на котором находится пункт меню, соответствующий часто используемой команде. С этой целью необходимо поместить ссылки на часто активируемые пункты меню на вышележащем уровне. Если ссылка также часто активируется, то следует создать ссылку на еще более высоком уровне и т.д.

2. Уменьшение количества пунктов меню: а) скрывать редко используемые ссылки на пункты меню (в Windows скрытые элементы можно увидеть при нажатии на кнопку раскрытия скрытой части подменю (рис. 3)); б) уничтожать редко используемые ссылки на пункты меню.

Предлагается пункты меню с большим количеством активаций дублировать на более высоком уровне с помощью разработанного алгоритма.

В общем случае адаптация меню для пользователя u состоит в перераспределении элементов меню по уровням иерархии его навигационной структуры и в изменении видимости (“виден”/”скрыт”) элементов меню.

Входными данными алгоритма являются: модель меню программного приложения; модель пользователя программного приложения; когнитивная константа C; K – количество активаций пункта меню, при котором необходимо создать ссылку; $V_{\min}$ – минимальное число активаций

пункта меню (пункт меню, для которого этот предел не достигнут, скрывается); T – минимальное время существования ссылки, ранее которого она не может быть удалена.

Алгоритм подстройки меню программного приложения для текущего пользователя при активации пункта меню $r \in R$ включает следующие шаги:

1. Если пункт меню r или подменю s, в котором находится пункт меню r, невидимы, то сделать их видимыми: $h^r = visible$, $h^s = visible$.

2. Если количество активаций пункта меню r больше величины $K(v^r > K)$, то перейти к шагу 3, иначе перейти к шагу 5.

3. Определить, есть ли ссылка на пункт меню r на вышестоящем уровне в подменю $\overline{s} = (s \in S \mid s \prec r, e \to r, l^r - l^s = 2)$. Если ссылка $\overline{e} = (e \in E \mid \overline{s} \prec e, l^e - l^{\overline{s}} = 1, e \to r)$ на пункт меню r в подменю $\overline{s}$ имеется $(\overline{e} \neq \varnothing)$, то $t^{\overline{e}} = t_c$, $v^{\overline{e}} = 0$, где t_c – текущее время, и закончить алгоритм.

4. Если $\overline{e} = \varnothing$, то создать ссылку e на пункт меню r:

$$e = (r, (s \in S \mid s \prec r, l^r - l^s = 2), 0, t_c)$$

и занести e в множество E: $E = E \cup e$.

5. Если количество ссылок в подменю s больше заданного предельного значения C, т.е. $|e \in E \mid s \prec e, l^e - l^s = 1| > C$, то необходимо удалить ссылку $\overline{e}$ из подменю s, у которой время существования превышает указанный срок T и за контролируемый период количество активаций минимально:

$$\overline{e} = (e \in E \mid s \prec e, l^e - l^s = 1, t_c - t^e > T, v^e = \min_e v^e)$$

$$E = E \backslash \overline{e}.$$

6. Если в множестве E есть ссылки E' с количеством активаций меньше $V_{\min}$, и со сроком существования более T, то необходимо их удалить:

$$E' = \{e \in E \mid t_c - t^e > T, v^e < V_{\min}\},$$

$$E = E \backslash E'.$$

7. Если есть пункты меню $r \in R$ с количеством активаций меньше $V_{\min}$, то необходимо их скрыть:

$$R' = \{r \in R \mid v^r < V_{\min}\},$$

$$h^r = hidden \ \forall r \in R'.$$

8. Если есть подменю, не содержащие других подменю, а включающие только элементы, активирующие команды, и количество активаций всех элементов указанных подменю меньше $V_{\min}$, то данные подменю необходимо скрыть:

$$S' = S\{s \in S \mid s \prec s', s' \in S\},$$

$$S'' = S' \backslash \{s \in S' \mid s \prec r \in R, v^r \geq V_{\min}\},$$

$$h^s = hidden \ \forall s \in S''.$$

Конец алгоритма.

Алгоритм адаптивной настройки меню программного приложения для текущего пользователя при активации ссылки e на пункт меню r отличается от предыдущего алгоритма шагами 1–3:

1. Если количество активаций ссылки e на пункт меню r больше величины $K(v^e > K)$, то перейти к шагу 2, иначе перейти к шагу 4.

2. Определить, есть ли ссылка на пункт меню r на вышестоящем уровне в подменю $\overline{s} = (s \in S \mid e \to r, e \in E, r \in R, s \prec e, l^e - l^s = 2)$.

3. Если ссылка $\overline{e} = (e \in E \mid \overline{s} \prec e, l^e - l^{\overline{s}} = 1, e \to r)$ на пункт меню r в подменю $\overline{s}$ имеется $(\overline{e} \neq \varnothing)$, то $t^{\overline{e}} = t_c$, $v^{\overline{e}} = 0$ и закончить алгоритм.

Описанные алгоритмы подстройки меню программного приложения для пользователя u предлагается выполнять при каждой активации пункта меню или ссылки на пункт меню.

Рассмотренные алгоритмы можно адаптировать таким образом, чтобы они реализовывались не при каждой активации пункта меню, а при запуске программы или по таймеру.

Помимо выше описанных алгоритмов при каждом запуске программы осуществляется определение пунктов и подпунктов меню, которые необходимо скрыть. Скрываются пункты меню, которые не активировались в течение времени T и пункты подменю, в которых отсутствуют подменю и каждый из находящихся в них пунктов меню не активировался:

$$h^r = hidden \ \forall r \in R \mid v^r = 0;$$

$$h^s = hidden \ \forall s \in S \mid s \in r, \quad r \in R, \quad \sum_{r \succ s} v^r = 0.$$

Непредсказуемая адаптация интерфейса может снизить удобство использования программного приложения, вызвать эмоциональную напряжен-

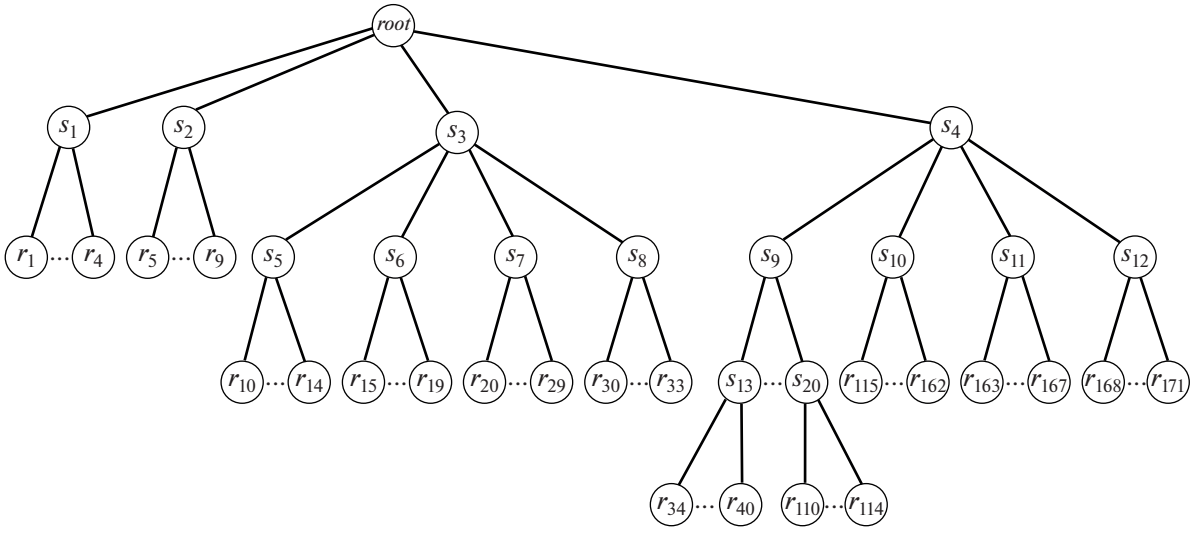


Рис. 4. Структура меню программного обеспечения до адаптивной подстройки.

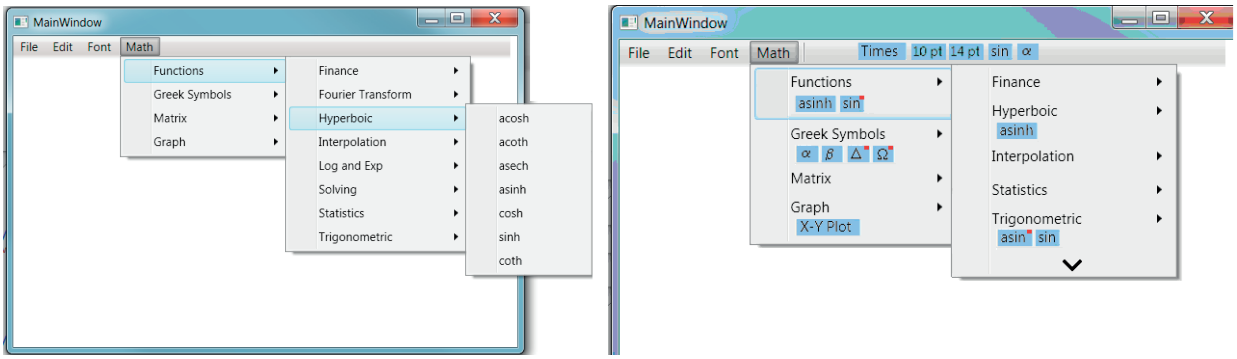


Рис. 5. Внешний вид меню программного обеспечения без ссылок и со ссылками.

ность пользователя и негативное восприятие неконтролируемых пользователем изменений интерфейса [5]. Для предотвращения этих проблем и повышения эргономичности меню программного приложения пользователю дается дополнительная возможность редактирования меню с учетом собственных предпочтений. Операции ручной подстройки пунктов меню активируются достаточно просто в режиме эксплуатации программного приложения без активации режимов настройки. Редактирование меню осуществляется устройством типа “мышь” при одновременном нажатии сочетания клавиш клавиатуры Ctrl/Shift/Alt. При этом пользователю предоставляется возможность:

1) выносить пункт меню или ссылку на пункт меню на уровень выше;

2) удалять ссылку на пункт меню;

3) менять положение пункта меню или ссылки на пункт меню, оставляя его на том же уровне;

4) делать пункт меню “видимым” или “невидимым”.

Изменения, внесенные пользователем, могут отменяться разработанным алгоритмом, но с согласия пользователя.

5. ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ АЛГОРИТМА

Тестирование разработанного алгоритма проводилось на тестовом программном обеспечении. Оно предназначено для набора форматированно-

Таблица 1. Результаты тестирования алгоритма

Пункт меню	Уровень пункта меню	Количество пунктов по уровням от корня дерева без учета корня	Количество активаций пунктов меню	Среднее время активации пунктов меню при тестировании 1, с	Общее среднее время активации пунктов меню при тестировании 1, с	Уровни ссылок на пункты меню	Среднее время активации пунктов меню и ссылок на них при тестировании 2, с	Среднее время активации пунктов меню и ссылок на них при тестировании 3, с
1	2	3	4	5	6	7	8	9
r_1	3	8(4+4)	3	1.54	4.32	–	1.52	1.54
r_2	3	8(4+4)	12	1.66	19.92	2	1.22	1.20
r_3	3	8(4+4)	67	1.62	108.54	2, 1	0.45	0.43
r_4	3	8(4+4)	1	1.63	1.53	–	1.61	1.60
r_9	3	9(4+4)	4	1.98	7.92	–	1.92	1.87
r_{11}	4	13(4+4+5)	14	2.81	51.8	3, 2	2.32	2.23
r_{12}	4	13(4+4+5)	1	2.82	3.8	–	2.81	2.71
r_{16}	4	13(4+4+5)	22	2.85	72.6	3, 2	2.21	2.10
r_{18}	4	13(4+4+5)	21	2.82	71.4	3, 2	2.24	2.22
r_{30}	4	12(4+4+4)	5	2.81	16.2	3	2.96	2.82
r_{31}	4	12(4+4+4)	2	2.88	6.46	–	2.87	2.72
r_{34}	5	23(4+4+8+7)	5	3.73	18.65	–	3.70	3.23
r_{35}	5	23(4+4+8+7)	6	3.80	22.8	4	3.68	3.57
r_{36}	5	23(4+4+8+7)	10	3.77	37.7	4, 3	3.40	3.28
r_{39}	5	23(4+4+8+7)	13	3.69	47.97	4, 3	3.31	3.12
r_{40}	5	23(4+4+8+7)	15	3.72	61.5	4, 3	3.25	3.10
r_{96}	5	20(4+4+8+4)	33	3.68	118.8	4, 3, 2, 1	0.96	0.95
r_{98}	5	19(4+4+8+3)	29	3.61	89.9	4, 3, 2, 1	1.25	1.22
r_{102}	5	19(4+4+8+3)	12	3.69	34.8	4, 3	2.51	2.33
r_{110}	5	24(4+4+8+8)	10	3.71	41	4, 3	2.42	2.31
r_{101}	5	24(4+4+8+8)	12	3.72	50.64	4, 3	2.43	3.33
r_{112}	5	24(4+4+8+8)	11	3.87	45.87	4, 3	2.39	2.34
r_{115}	4	56(4+4+48)	30	3.22	96.6	3, 2, 1	1.29	1.23
r_{116}	4	56(4+4+48)	30	3.41	102.3	3, 2, 1	1.31	1.28
r_{118}	4	56(4+4+48)	3	3.43	10.29	–	3.45	3.44
r_{138}	4	56(4+4+48)	12	3.24	38.88	3, 2	2.41	2.21
r_{142}	4	56(4+4+48)	4	3.25	13	–	3.23	3.23
r_{162}	4	56(4+4+48)	22	3.36	73.92	3, 2, 1	1.63	1.57
r_{168}	4	12(4+4+4)	7	3.2	22.4	3	3.01	3.00

го текста и математических формул, выбираемых из пунктов главного меню. Структура меню представлена на рис. 4.

Меню программного обеспечения содержит 171 элемент, каждый из которых соответствует пункту меню, активирующему команду. Иерархическая структура меню представляется деревом с глубиной, равной 5. Внешний вид меню программы до и после создания ссылок представлен на рис. 5. Выделение цветом ссылок позволяет визуально отделить их от пунктов меню и тем самым время доступа к пунктам меню и ссылкам практически не увеличивается из-за суммарного увеличения количества пунктов меню и ссылок. Таким образом, в каждом подменю $s \in S$ достаточно обеспечить $|r \in R \mid r \succ s \in S, l^r - l^s = 1| \le C$ и $|e \in E \mid e \succ s \in S, l^e - l^s = 1| \le C$. Кроме того, недавно созданные ссылки на пункты меню, которые активировались не более двух раз, дополнительно выделяются меткой (в правом верхнем углу ссылок на рис. 5), а пункты меню, которые не активировались дольше времени T, и подпункты меню, в которых каждый пункт меню не активировался ни разу, скрываются.

Для оценки эффективности предлагаемого алгоритма пользователям в количестве 36 человек было выдано задание на ввод в программу текста за минимально возможное время. По мере набора текста необходимо было активировать пункты меню, указанные в задании. В ходе тестирования пользователи должны были выполнить 416 активаций 29 пунктов меню, описанных в столбцах 1–4 табл. 1. В табл. 1 приведены только пункты меню, активированные хотя бы один раз. Ошибочно активированные пользователями пункты меню в таблицу не включены.

Для оценки эффективности предложенных алгоритмов было проведено три тестирования с интервалом в одну неделю с привлечением к тестированию одних и тех же пользователей.

Тестирование 1 проводилось без создания ссылок на пункты меню. Всего на выполнение задания было потрачено в среднем 2642 с. Общее время активации всех указанных в задании пунктов меню составило в среднем на одного пользователя 1261.55 с. Результаты тестирования 1 приведены в столбцах 5, 6 табл. 1.

Тестирование 2 было проведено с активацией алгоритма для создания ссылок на пункты меню. Общее время тестирования было уменьшено до 2127 с в среднем на одного пользователя, т.е. на 19.5%. В результате время активации пунктов меню и ссылок на них составило 741.31 с, что сэкономило 4123% времени активации пунктов меню и ссылок на них. Уровни ссылок на пункты меню и результаты тестирования 2 приведены в столбцах 7, 8 табл. 1.

Тестирование 3 проводилось с активацией алгоритма с созданием ссылок на пункты меню и предварительным скрытием некоторого количества заведомо ненужных пунктов меню. В результате время активации пунктов меню уменьшилось до 724.72 с в среднем на одного пользователя, т.е. на 42.55%. Результаты тестирования приведены в столбце 9 табл. 1.

6. ЗАКЛЮЧЕНИЕ

В статье предложен подход к адаптации меню программного приложения к потребностям конкретного пользователя посредством добавления ссылок на часто используемые пункты меню на более высоких уровнях его навигационной структуры. Результаты свидетельствуют о получении практического эффекта в виде уменьшения времени на активацию пунктов меню. Разработанный алгоритм и его программная реализация позволяют создать программное приложение с адаптивным меню, повышающим эффективность работы пользователя, заключающуюся в экономии времени обращения к пунктам меню.

Предлагаемый в данной статье подход к построению адаптивного меню характеризуется следующими особенностями.

1. На часто используемые элементы меню создаются ссылки, расположенные на уровень выше, вместо переноса этих элементов в начало списка на том же уровне в отличие от [8].

2. Ссылки выделяются визуально, что позволяет рассматривать их в качестве “параллельного” набора команд и независимо применять к этому набору накладываемое когнитивной константой ограничение.

3. Наряду с автоматической адаптацией предложены способы ручной настройки меню пользователем без необходимости входа в специальный режим настройки. Данное решение представляет комбинацию свойств адаптивных и адаптируемых меню.

Дальнейшее исследование предполагает изучение возможности повышения эффективности работы пользователя программного приложения за счет повышения адаптационных свойств других элементов пользовательского интерфейса.

СПИСОК ЛИТЕРАТУРЫ

1. *López-Jaquero V., Montero F., Molina J.P., Fernández-Caballero A., González P.* Model-Based Design of Adaptive User Interfaces through Connectors // Interactive Systems. Design, Specification, and Verification (DSV-IS 2003). Lecture Notes in Computer Science, Springer, Berlin, Heidelberg. 2003. V. 2844. P. 245–257.
2. *Paymans T.F., Lindenberg J., Neerincx M.* Usability trade-offs for adaptive user interfaces: Ease of use and learnability // Proceedings of the 9th International Conference on Intelligent User Interfaces (IUI '04). 2004. P. 301–303.
3. *Peissner M., Schuller A., Spath D.* A Design Patterns Approach to Adaptive User Interfaces for Users with Special Needs // Human-Computer Interaction. Design and Development Approaches (HCI 2011). Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. 2011. V. 6761. P. 268–277.
4. *Zosimov V.V., Khrystodorov O.V. & Bulgakova O.S.* Dynamically Changing User Interfaces: Software Solutions Based on Automatically Collected User Information // Programming and Computer Software. 2018. V. 44. P. 492–498.
5. *Takacs B., Simon L.* Sensing User Needs: Recognition Technologies and User Models for Adaptive User Interfaces // Human-Computer Interaction. Design and Development Approaches (HCI 2011). Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. 2011. V. 6761. P. 498–506.
6. *Nazemi K., Stab C., Fellner D.W.* Interaction Analysis for Adaptive User Interfaces // Advanced Intelligent Computing Theories and Applications (ICIC 2010). 2010. V. 6215. P. 362–371.
7. *Alvarez-Cortes V., Zarate V.H., Uresti J.A.R., Zayas B.E.* Current Challenges and Applications for Adaptive User Interfaces // Human-Computer Interaction. 2009. P. 13–30.
8. *Sears A., Shneiderman B.* Split menus: Effectively using selection frequency to organize menus // ACM Transactions on Computer-Human Interaction. 1994. V. 1. № 5. P. 27–51.
9. *Soui M., Chouchane M., Mkaouer M.W., Kessentini M., Ghedira K.* Assessing the quality of mobile graphical user interfaces using multi-objective optimization // Soft Computing. 2019. P. 1–30.
10. *Sluis-Thiescheffer R.J.W., Bekker M.M., Eggen J.H., Vermeeren A.P.O.S., De Ridder H.* Development and application of a framework for comparing early design methods for young children // Interacting with Computers. 2011. V. 23. № 1. P. 70–84.
11. *Akiki P.A., Bandara A.K., Yu Y.* Adaptive Model-Driven User Interface Development Systems // ACM Computing Surveys (CSUR). 2014. V. 47. № 1. Article no 9. 33 p.
12. *Findlater L., McGrenere J.* A Comparison of Static, Adaptive, and Adaptable Menus // ACM Conference on Human Factors in Computing (CHI 2004). 2004. V. 6. № 1. P. 89–96.
13. *Park J., Han S.H.* Integration of Adaptable and Adaptive Approaches for Interface Personalization through Collaborative Menu // International Journal of Human-Computer Interaction. 2012. V. 28. № 9. P. 613–626.
14. *Elio R., Haddadi A.* Dialog management for an adaptive database assistant (Technical Report 98–3). Daimler-Benz Research and Technology Center, Palo Alto, CA. 1998.
15. *Zhang L., Qu Q.-X., Chao W.-Y., Duffy V.G.* Investigating the combination of adaptive UIs and adaptable UIs for improving usability and user performance of complex UIs // International Journal of Human-Computer Interaction. 2020. V. 36. № 1. P. 82–94.
16. *Hervás R., Bravo J.* Towards the ubiquitous visualization: Adaptive user-interfaces based on the Semantic Web // Interacting with Computers. 2011. V. 23. № 1. P. 40–56.
17. *Park K., Lee S.W.* Model-Based Approach for Engineering Adaptive User Interface Requirements // Requirements Engineering in the Big Data Era. Communications in Computer and Information Science. Springer, Berlin, Heidelberg. 2015. V. 558. P. 18–32.
18. *Schlimmer J.C., Hermens L.A.* Software Agents: Completing Patterns and Constructing User Interfaces // Journal of Artificial Intelligence Research. 1993. V. 1. P. 61–89.
19. *Mitchell J., Shneiderman B.* Dynamic versus static menus: an exploratory comparison // SIGCHI Bull. 1989. V. 20. № 4. P. 33–37.
20. *McGrenere J., Baecker R., Booth K.* An evaluation of a multiple interface design solution for bloated software // Processing of the SIGCHI Conference on Human Factors in Computing Systems (CHI '02). Association for Computing Machinery, New York, NY, USA. 2002. V. 4. № 1. P. 164–170.
21. *Shneiderman B.* Direct manipulation for comprehensible, predictable and controllable user interfaces // Intelligent User Interfaces (IUI '97). 1997. P. 33–39.
22. *Matsui S., Yamada S.* Optimizing hierarchical menus by genetic algorithm and simulated annealing // Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08). 2008. P. 1587–1594.

23. *Abascal J., Aizpurua A., Cearreta I., Gamecho B., Garay N., Miñón R.* Some Issues Regarding the Design of Adaptive Interface Generation Systems // Proceedings of the 6th International Conference on Universal Access in Human-Computer Interaction. Design for All and inclusion (UAHCI 2011). Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg. 2011. V. 6765. P. 307–316.

24. *Langley P.* Machine learning for adaptive user interfaces // KI-97: Advances in Artificial Intelligence. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. 1997. V. 1303. P. 53–62.

ПРОГРАММИРОВАНИЕ, 2020, № 6, с. 41–54

ПАРАЛЛЕЛЬНОЕ И РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ

УДК 004.92+004.94

ПРОГРАММНАЯ СИСТЕМА ОБРАБОТКИ ИЗОБРАЖЕНИЙ С ПАРАЛЛЕЛЬНЫМИ ВЫЧИСЛЕНИЯМИ

© 2020 г. К. И. Кий[a],*, Д. А. Анохин[a],, А. В. Подопросветов[a],*****

[a] *Институт прикладной математики им. М.В. Келдыша РАН, Москва, Миусская пл., д. 4, 125047 Россия*

**E-mail: kikip_46@mail.ru*

***E-mail: anodmitry@gmail.com*

****E-mail: llecxis@gmail.com*

Поступила в редакцию 17.12.2019 г.
После доработки 05.02.2020 г.
Принята к публикации 11.04.2020 г.

В работе описывается программная система обработки изображений с параллельными вычислениями, основанная на методе геометризованных гистограмм, разработанным для содержательного описания и сегментации цветных изображений и для создания систем понимания изображений реального времени. Параллельная обработка опирается на то, что в отличии от большинства существующих методов сегментации изображений, указанный метод построен так, что наиболее трудоемкие операции работы с пиксельным массивом могут выполняться в нем при помощи n независимых потоков. Описываются принципы построения программ обработки данных в отдельных потоках, в которых создается содержательное сжатое описание изображения (кадра видеопоследовательности), сохраняющее геометрические связи исходного изображения, но имеющее размерность на несколько порядков меньше, чем у него. Основные операции сегментации и систем понимания изображений выполняются без использования пиксельного массива изображения – с построенным сжатым содержательным описанием. На эти операции требуется малое (в среднем менее 10 мс на кадр для всей совокупности задач) время исполнения на стандартных современных персональных компьютерах, даже для видео высокого разрешения. В работе описана многопоточная реализация создания сжатого описания изображения (кадра), которая позволяет довести и без того быструю работу системы до рекордных образцов быстродействия. Также описываются применения к системам понимания дорожных сцен, таким как системы нахождения области дороги и ее окрестности, области неба и детектирования и понимания дорожной разметки (постоянной белой и цветной, временной), а также к поиску сигнальных огней вертолетов. Приводятся и обсуждаются примеры обработки конкретных дорожных сцен и даются оценки быстродействия системы на видеопоследовательностях реальных дорожных сцен.

DOI: 10.31857/S0132347420060047

1. ВВЕДЕНИЕ

Технологии обработки изображений для обнаружения объектов или событий часто требуют обработку потока кадров. Вследствие этого появляется множество подходов к обработке видеорядов, в том числе в реальном времени. Их можно разделить как по основным целям, например: автономное вождение, медицинское применение, системы безопасности, отслеживание и сортировка объектов, так и по методам и подходам. Классические методы сегментации [1], методы многоклассовой сегментации, основанные на обучении [2, 3] и методы, основанные на нейросетях [3, 4], дают несколько подходов к решению задач сегментации и большого круга задач понимания изображений в реальном времени. Данные методы в качестве существенного этапа содержат некоторую процедуру кластеризации в пространстве признаков. Это автоматически приводит к потере малых объектов на изображении. Необходимо также отметить, что при определении близости значений признаков не используется информация о локализации пикселов, в которых принимаются данные значения признаков. Это не позволяет работать с изображениями с сильно неоднородным освещением. Следует также отметить, что возможность распараллеливания не предусмотрена исходно в методах [1–3]. Также важно, что для решения задач управления объектами с обратной связью по зрению, сама пиксельная сегментация является лишь промежуточным результатом, и требуется дополнительная боль-

шая работа, чтобы получить данные для управления.

Кроме того, имеется много подходов, ориентированных на конкретные специфические задачи, такие как, например, задача распознавания граничных разметок полосы движения автомобиля [4, 5]. Подробный обзор таких методов содержится в [6]. Также данной проблеме посвящена [7], и она содержит полезную библиографию по данной проблеме. У всех вышеперечисленных подходов есть существенные ограничения на область применения, поскольку необходимые вычислительные мощности, обеспечивающие достаточную точность, часто превышают рамки ресурсов, выделенных под задачу. Это обусловлено большим объемом информации, получаемой из видеопоследовательностей и, как следствие, сложностью алгоритмов обработки. Для ускорения обработки видеоряда часто используются многопоточные алгоритмы, выполняемые как на CPU, так и на GPU [8, 9]. Данный подход широко применим, поскольку алгоритмы обработки, работающие с матрицами пикселов, могут быть преобразованы в многопоточные алгоритмы не только на уровне языка программирования, но и на функциональном уровне. Благодаря развитию современных компьютерных архитектур, параллельное программирование является областью науки с большим потенциалом. Использование технологий многопоточного программирования может значительно сократить время расчетов, ускорив достижение основной цели программы.

Дополнительные возможности построения эффективных систем понимания изображений обеспечивает подход, при котором алгоритмы решения задачи сегментации заранее подразумевают возможность их дальнейшей адаптации для работы в многопроцессорных системах, а результаты сегментации записаны в сжатом виде, непосредственно подходящем для практического применения в задачах, использующих компьютерное зрение. Такими свойствами обладает метод геометризованных гистограмм, разработанный в [10–12] и обрабатывающий изображение как в черно-белом, так и в цветном форматах. В данной работе будет рассматриваться этот метод, его программная реализация, а также его адаптация для многопоточного исполнения на CPU. Данный метод является достаточно гибким и позволяет решать как задачи обнаружения и слежения за объектами, так и задачи распознавания. С помощью метода геометризованных гистограмм удается находить на изображениях и кадрах видеопоследовательностей в реальном времени как большие, так и малые объекты: цветные метки (движущиеся и неподвижные), дорогу, обочины, сигнальные огни автомобилей и вертолетов, дорожную разметку (временную и постоянную), дорожные знаки и сигналы светофоров [11–14]. Все эти задачи можно решать в реальном времени для видео высокого разрешения (1280 × 720 пикселов и более) на основе сжатого описания изображений. В разделе 2 будет дано описание метода геометризованных гистограмм и его программной реализации. В разделе 3 будет описано применение метода многопоточной обработки для реализации сжатого описания данных видеоряда по методу геометризованных гистограмм. В разделе 4 будет дана блок-схема полученной системы обработки, сегментации и понимания изображений в реальном времени. Также будут описаны применения разработанной системы, даны результаты обработки видеопоследовательностей в реальном времени и приведены результаты оценки быстродействия системы на реальных видеопоследовательностях дорожных сцен.

2. ОСНОВНЫЕ ПРИНЦИПЫ МЕТОДА ГЕОМЕТРИЗОВАННЫХ ГИСТОГРАММ

В отличие от основных методов сегментации изображений, метод геометризованных гистограмм построен так, что основная обработка видеоданных будет вестись параллельным образом. В основе метода геометризованных гистограмм лежит основная идея математической редукции сложных задач. Эта идея предполагает введение новых понятий и логических систем, таких, что вложение в эти системы исходных задач и переформулирование их в новых терминах делает их разрешимыми с помощью новой техники (современная теоретическая математика) или дает эффективные методы их численного решения (вычислительные науки и программирование). Так как пиксельная сегментация является лишь промежуточным продуктом в построении систем понимания изображений и плохо годится для решения задач управления с обратной связью по зрению, то в методе геометризованных гистограмм она в основном не используется.

2.1. Конструкция геометризованных гистограмм

С помощью техники, описанной в [10–12], каждому цветному изображению ставится в соответствие структурный граф цветовых сгустков *STG*. Чтобы построить *STG*, изображение разбивается на полосы одинаковой ширины St_n со сторонами, параллельными горизонтальной или вертикальной оси плоскости изображения *Os*. Вводится понятие геометризованной гистограммы изображения [10], которая является далеким обобщением обычной гистограммы. Однако обычная гистограмма является слабым инвариантом изображения, так

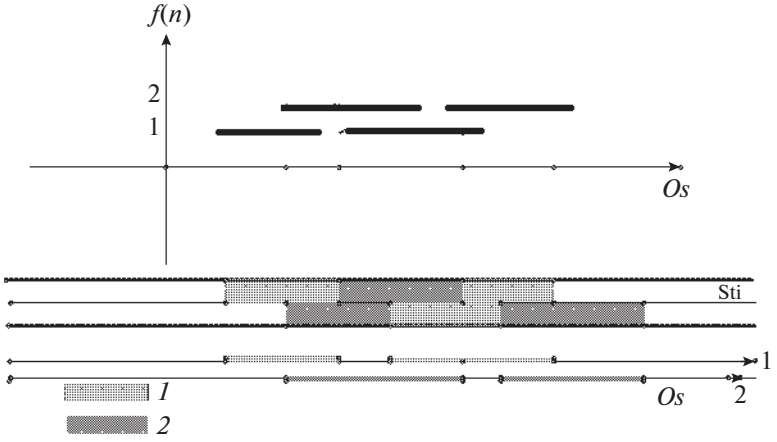


Рис. 1. Получение интервалов геометризованной гистограммы для модельного изображения.

как она остается инвариантной при любом взаимно-однозначном преобразовании прямоугольника изображения. В ней совершенно не учитывается геометрия объектов изображения. В то время как геометризованная гистограмма остается инвариантной только относительно преобразований внутри полос, при которых точки передвигаются перпендикулярно оси *Os*. Так как мы имеем дело с узкими полосами, эти преобразования почти не меняют геометрии объектов, принадлежащих изображению.

Геометризованная гистограмма достаточно точно описывает распределение значений функции, задающей изображение в прямоугольнике кадра. Получается геометризованная гистограмма с помощью проектирования пикселов полосы на ее нижнее основание. Мы будем обозначать подмножества полосы St_n, в которых функция, задающая изображение, принимает фиксированное значение $z - L_z$ и называть их множествами уровня z. Для монохромного изображения [10], ввиду дискретного характера изображений, проекция любого L_z, на нижнее основание полосы есть объединение интервалов $Pr(L_z) = \cup_k I_{kz}$ на нижней оси полосы.

Заметим, что описание геометрии множеств L_z на плоскости изображения есть описание структуры распределения значений функции $f(x, y)$, задающей изображение. Описание этой геометрии на всей плоскости носит сложный характер. Использование в задачах сегментации описания распределения значений функции f выведет решение задачи о сегментации изображения на новый уровень по сравнению с изучением частотных закономерностей в области ее значений. Задача приближенного описания множеств L_z в узких полосах легко поддается решению. Это решение дается геометризованной гистограммой полосы. Чтобы сравнивать множество уровней в разных полосах, можно считать, что все интервалы лежат на оси *Os*. Рисунок 1 иллюстрирует получение интервалов геометризованной гистограммы в полосе. В полосе задана функция изображения с двумя значениями, которая постоянна в выделенных прямоугольниках. Под полосой расположены два экземпляра оси *Os*. На них показаны системы интервалов, соответствующие областям одинакового значения функции изображения. На самом деле, эти системы интервалов лежат на одной оси. Два экземпляра взяты для наглядности. При проектировании для каждого интервала *Sg* вычисляются его границы beg_{Sg} и end_{Sg} на оси *Os*, а также его мощность $Card^{Sg}$ – количество точек полосы, в которых функция изображения принимает заданное значение и которые проектируются на данный интервал. Объединение систем интервалов по всем полосам хорошо описывает распределение значений монохромной функции, задающей изображение. Эта конструкция обобщается на случай векторной функции, задающей цветное изображение [10]. Удается при проектировании разделить все множество точек полосы на подмножества, в которых насыщение, оттенок, и яркость варьируются в некоторых диапазонах. Эти подмножества проектируются на интервалы на оси *Os*. В каждой полосе получаются системы интервалов, каждый из которых (*Sg*) характеризуется следующими параметрами:

- Положение интервала $[beg_{Sg}, end_{Sg}]$ *Sg* на оси *Os*.
- Диапазон $\Delta H^{Sg} = [H^{Sg}_{\min}, H^{Sg}_{\max}]$ и среднее значение оттенка H^{Sg}_{mean}.
- Диапазон $\Delta S^{Sg} = [S^{Sg}_{\min}, S^{Sg}_{\max}]$ и среднее значение S^{Sg}_{mean} насыщения.

• Диапазон $\Delta I^{Sg} = [I^{Sg}_{\min}, I^{Sg}_{\max}]$ и среднее значение яркости I^{Sg}_{mean}.

• Мощность интервала $Card^{Sg}$ (приблизительно равно числу точек полосы, лежащих в полосе над интервалом, с цветовыми характеристиками, заключенными в интервалах, указанными выше для *Sg* на оси *Os*).

Обозначим $dens(Sg) = Card^{Sg}/(end_{Sg} - end_{Sg} + 1)$ плотность интервала *Sg*. Естественно, если изображение в полосе состоит из однородно закрашенных цветовых пятен с фиксированными цветовыми характеристики, то подобно рис. 1, каждому такому цветовому пятну при проектировании будет соответствовать интервал геометризованной гистограммы с теми же фиксированными цветовыми характеристиками. Однако изображения сцен реального мира с тенями, областями неоднородного освещения и влажными частями намного сложнее. Геометризованные гистограммы полос таких изображений содержат сложные системы пересекающихся интервалов с заданными диапазонами и средними значениями яркостно-цветовых характеристик. Для того, чтобы давать семантическое описание изображений и решать задачи их понимания на основе геометризованных гистограмм, требуется введение большого числа новых понятий и аксиоматизированных конструкций; также необходимо разработать эффективные программные средства для построения объектов, удовлетворяющих постулированным свойствам и их интерпретации.

Существенным свойством алгоритма получения геометризованной гистограммы в полосе является то, что она может быть получена за один проход массива точек полосы. Это обеспечивает ее получение в реальном времени. Также это является ключевым моментом параллельной реализации метода. Даже для монохромной функции изображения, построение геометризованной гистограммы за один проход изображения является занятной программистской задачей, которая может предлагаться на олимпиадах по программированию. Она реализуется кодом, объем которого более 300 строк. Код программы построения геометризованной гистограммы монохромной функции изображения выложен на [18], поэтому ввиду размера код в статье не приводится. Построение геометризованной гистограммы цветного изображения за один его проход является существенно более сложной программистской задачей. Код ее реализации содержит более 2000 строк. Существенное место в коде занимает сортировка пикселов согласно с их яркостно-цветовыми характеристиками. При сортировке мы используем цветовые координаты ($G/(G + B)$, $G/(G + R)$, I), введенные первым автором в [10]. Здесь R, G, B – обычные цветовые координаты, а I – полутоновая интенсивность. Введем характеристическую функцию *CF*. Заметим, что функции $G/(G + B)$, $G/(G + R)$ задают цветовые координаты на цветовом треугольнике [10]. Если оттенок пиксела принадлежит желтой части цветового треугольника, тогда *CF* совпадает с $G/(G + B)$. Когда мы движемся в следующий цветовой диапазон (зеленый, голубой, красный), значение $G/(G + B)$ сдвигается на M, где M – число градаций функции $G/(G + B)$. Геометризованная гистограмма *CF*, дополненная для каждого ее интервала I_{kz} классической гистограммой другой цветовой координаты $G/(G + R)$ и диапазоном, и средним значением полутоновой компоненты спроектированных на интервал пикселов, называется геометризованной гистограммой цветного изображения в полосе St_i. На основе полученных данных для каждого интервала вычисляются диапазоны и средние значения координат H (оттенок), S (насыщение), I (полутоновая интенсивность). Далее характеристики интервалов применяются в этой цветовой системе координат.

Пример геометризованной гистограммы выделенной полосы приведен на рис. 2. Точка в правой части изображения, расположенного в верхней части рисунка, выделяет горизонтальную его полосу. В нижней части рисунка показаны все интервалы геометризованной гистограммы этой полосы. По горизонтальной оси откладываются значения характеристической функции геометризованной гистограммы *CF*. По вертикальной оси откладываются координаты концов интервалов (измеренные по горизонтальной оси изображения). Овалами (при движении слева направо) выделены интервалы геометризованной гистограммы, соответствующие: 1) частям белого автомобиля между габаритами и зонами сигналов торможения; 2) частям зоны стоп-сигналов на белом автомобиле, которые имеют красный цвет; и 3) частям соответствующих зон растительности на обочинах (верхний и нижний овалы). Квадраты в верхней части визуализации геометризованной гистограммы показывают цвет того из интервалов, определяемых стоп-сигналами, к которому подведена стрелка мыши.

Так как мы имеем дело с реальными объектами, то им соответствуют наборы интервалов геометризованной гистограммы, яркостно-цветовые характеристики которых варьируются. На сайте [18] выложена программа с инструкциями, которая позволяет построить и визуализировать, как на рис. 2, геометризованную гистограмму любого изображения в формате BMP.

Программная реализация построения геометризованных гистограмм цветного изображения

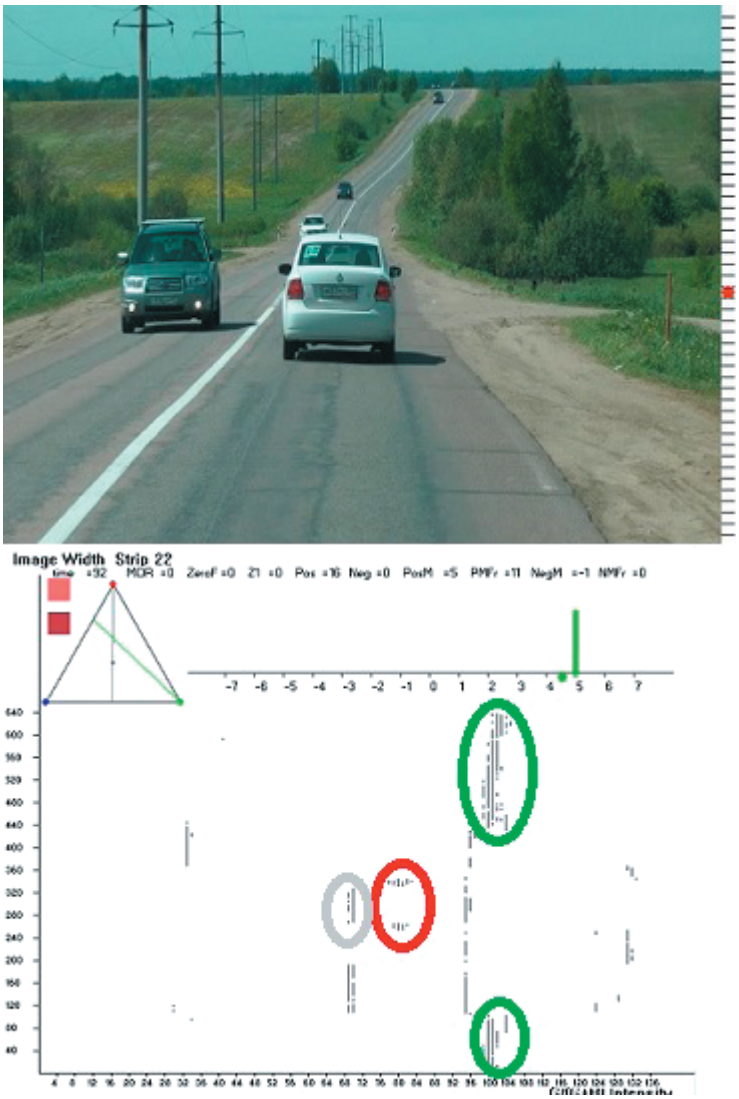


Рис. 2. Геометризованная гистограмма выделенной полосы.

(кадра видеопоследовательности) и его полутоновой компоненты находится в Блоке 1 итоговой блок-схемы программной системы из раздела 4. Для реальных изображений это является правилом, что реальным объектам соответствуют несколько геометрически близких интервалов геометризованной гистограммы, у которых яркостно-цветовые характеристики имеют близкие значения. Объединение геометризованных гистограмм полос дает геометризованную гистограмму всего изображения.

2.2. Построение цветовых сгустков и графа STG

Для того чтобы выделить на геометризованной гистограмме полосы части реальных объектов, в ней необходимо применить некоторые процедуры для объединения ее интервалов, им соответствующим. С помощью оригинальной операции кластеризации [10] интервалы геометризованной гистограммы объединяются в кластеры – цветовые сгустки, которые характеризуются такими же, как и у интервалов, яркостно-цветовыми параметрами, мощностью и плотностью. На этом шаге проводится объединение интервалов геометризованной гистограммы, интервалы локализации которых имеют сильное пересечение [10].

Программная реализация этой процедуры разделяется на следующие шаги:

1. Все интервалы геометризованной гистограммы выбрасываются на три отрезка прямой, соответствующей средней линии полосы. На первом отрезке остаются номера интервалов максимальной плотности, на втором – интервалы наиболь-

Рис. 3. Цветовые сгустки изображения из рис. 2. Черточками выделены границы зеленой обочины, полученные с помощью программной системы.

шего локального насыщения, на третьем – интервалы максимального значения полутоновой компоненты.

2. Интервалы с номерами, “выжившими” на всех трех отрезках средней линии, являются зародышами, с которых стартует процедура кластеризации.

3. К каждому зародышу присоединяются интервалы гистограммы, которые имеют с ними сильное пересечение [10] и близкие яркостно-цветовые характеристики.

Далее работает процедура объединения цветовых сгустков. При этом объединяются цветовые сгустки, имеющие соседние интервалы локализации и сходные яркостно-цветовые параметры. Итоговые цветовые сгустки b (color bunches в английском варианте термина) характеризуются интегральными параметрами ΔH^b, H^b_{mean}, $\Delta S^b S^b_{mean}$, ΔI^b, I^b_{mean}, $Card^b$, получаемыми суммированием и усреднением соответствующих интервалов геометризованной гистограммы, из которых они состоят. Введем так же, как и для интервалов геометризованной гистограммы, плотность цветового сгустка как $dens(b) = Card^b/L([beg_b, end_b])$ (мощность, поделенная на длину интервала). Процедуры построения цветовых сгустков и их объединения принадлежат Блоку 2 программы.

Цветовые сгустки объединяются в граф. В полосе соединяются ребром соседние цветовые сгустки (с соседними интервалами локализации), а в соседних полосах – цветовые сгустки, интервалы локализации которых пересекаются. Неформально, каждый сгусток дает описание некоторой части реального объекта в полосе, его проекцию на ось Os и описание значений численных цветовых характеристик этой части объекта. STG можно интерпретировать геометрически с помощью наложения отрезков его сгустков ($[beg_b, end_b]$) на центральную линию соответствующей полосы и окрашивания этих отрезков в цвет, определяемый H^b_{mean}, S^b_{mean}, I^b_{mean}. Пример множества цветовых сгустков изображения с рис. 2, наложенных на его полутоновую компоненту, можно посмотреть на рис. 3. Примеры цветовых сгустков, наложенных на цветные изображения, можно также посмотреть в открытом доступе в [13, 14].

Цветовые сгустки являются аналогом суперпикселов, применяемых в классических методах сегментации. Также с помощью программы из [18] можно построить и визуализировать множество цветовых сгустков для любого изображения в формате BMP. Приведенные изображения цветовых сгустков и получаемые с помощью программы из [18] показывают, что граф цветовых сгустков хорошо описывает яркостно-цветовые свойства изображений и геометрию реальных объектов в полосах и в изображении в целом.

2.3. Построение поисковой решетки на STG

На множестве цветовых сгустков строится “решетка поиска” SearchLat(STG) [12], которая позволяет производить глобальный анализ изображения. Если мы положим на среднюю линию каждой полосы разбиения изображения интервалы геометрической локализации $[beg_b, end_b]$ всех цветовых сгустков полосы, то получим некоторое ее покрытие.

Определение 2.1. Цветовые сгустки, имеющие в некоторой точке средней линии максимальную плотность, называются доминирующими цветовыми сгустками. Остальные сгустки называются доминируемыми.

Ясно, что доминирующие цветовые сгустки образуют покрытие средней линии. Оказывается [12], что всегда можно выбрать линейно-упорядоченную последовательность базисных цветовых сгустков, которые образуют покрытие средней линии. Алгоритм для построения SearchLat(STG) состоит из следующих шагов:

1. Для каждого доминирующего сгустка находятся доминирующие сгустки слева и справа, интервалы локализации являются соседними для интервала локализации выбранного сгустка.

2. Начиная с левого конца, строится упорядоченная система доминирующих цветовых сгустков, соседних друг другу.

Доминирующие цветовые сгустки, включенные в линейно-упорядоченное покрытие, назы-

ваются базисными цветовыми сгустками. Базисные цветовые сгустки всех полос образуют решетку поиска изображения SearchLat(*STG*) [12].

В дополнение к поисковой решетке строится структура BunchLoc(*STG*). В каждой полосе для любой точки x оси Os данная структура указывает номера всех цветовых сгустков, которые проходят через данную точку. Программа, реализующая построение структуры BunchLoc(*STG*), реализуется в два шага:

1. Перебираются все цветовые сгустки, и для каждой точки $x \in Os$ определяется число сгустков, проходящих через данную точку. Вычисляется N_b – максимальное число сгустков, которые могут проходить через произвольную точку оси Os.

2. Строится массив, высота которого равна N_b и который содержит для каждой точки $x \in Os$ список цветовых сгустков, проходящих через эту точку.

Структура BunchLoc(*STG*) позволяет для каждого цветового сгустка b определить, какие цветовые сгустки в соседних полосах связаны с данным сгустком ребром в *STG* (имеют интервалы локализации, пересекающиеся с интервалом локализации b). Структуры SearchLat и BunchLoc позволяют находить все соседние цветовые сгустки для выбранного сгустка b. После выполнения операций сегментации, цветовым сгусткам присваиваются метки объектов, которым они принадлежат. Эти структуры позволяют строить отношения соседства для выделенных на изображении объектов и выполнять глобальный анализ их взаимодействия [12]. Их конструирование выполняется в Блоке 3.

2.4. Построение глобальных объектов

Чтобы решать задачи поиска объектов с заданными геометрией и яркостно-цветовыми характеристиками на изображениях и задачи понимания изображений, необходимо строить образы частей предполагаемых реальных глобальных объектов в *STG*. Под глобальными объектами мы понимаем объекты, расположенные более чем в одной полосе изображения. Реальные глобальные объекты могут быть двух типов. К первому типу относятся областные объекты. Они в полосах, как правило, при проектировании на *STG* индуцируют доминирующие цветовые сгустки. Глобальные объекты второго типа – это малые и тонкие объекты, такие как дорожная разметка, строительные конусы для обозначения запретных зон на дороге, знаки аварийной остановки, дорожные знаки, стоп-сигналы автомобилей и т.д. Объекты такого типа могут при проектировании давать как доминирующие (на близком расстоянии), так и доминируемые цветовые сгустки. Целый объект или его часть, лежащие в нескольких полосах, дают в некотором смысле непрерывные системы цветовых сгустков в *STG*. Для нахождения образов таких объектов, необходимо формализовать понятие непрерывной системы цветовых сгустков. Многие задачи по поиску ориентиров и объектов в кадре, можно переформулировать строго как задачи поиска некоторых непрерывных абстрактных объектов на графе цветовых сгустков.

Для этих целей в [11] введены понятия левых и правых контрастных кривых или левых и правых ростков глобальных объектов на *STG*, и определен двудольный граф левых и правых контрастных кривых *LRG*. Если изображение разбито на горизонтальные полосы, то, неформально, левая или правая контрастная кривая (росток глобального объекта) есть цепочка доминирующих цветовых сгустков в соседних полосах с подобными цветовыми характеристиками. При этом левые или правые концы цветовых сгустков меняются от полосы к полосе “непрерывно”, и соседние в той же полосе слева или справа цветовые сгустки имеют контрастные цветовые характеристики. Эта цепочка строится снизу вверх, переходя из полосы в полосу [11]. Для того, чтобы избежать операций полного перебора, непрерывное продолжение цветового сгустка в соседние полосы (построение левых и правых ростков глобальных объектов) осуществляется с использованием структуры BunchLoc. Конструкция левых и правых непрерывных цепочек контрастных доминирующих цветовых сгустков (левых и правых ростков глобальных контрастных объектов) выполняется в Блоке 4 программной системы. Также в этом блоке строится двудольный граф связи между левыми и правыми ростками глобальных объектов. Двудольный граф *LRG* показывает, какие левые и правые ростки в полосах можно соединить цепочками цветовых сгустков без контраста. Следовательно, эти ростки могут быть частями одного объекта. Для решения различных задач понимания изображений необходимо находить прямолинейные участки в граничных элементах построенных левых и правых ростков. Методы, основанные на методе наименьших квадратов, являются неустойчивыми относительно ошибок сегментации. Устойчивый метод, основанный на анализе гистограмм углов наклона отрезков, соединяющих концы интервалов локализации сгустков, входящих в построенный росток глобального объекта, разработан в [12]. Часть программы, реализующая предложенный метод, находится в Блоке 5 диаграммы. Этот блок заканчивает часть программного комплекса, который дает аппарат для решения различных задач понимания изображений. Общий объем кода больше 40000 строк. Предполагается

Рис. 4. Непрерывная слева система цветовых сгустков, соответствующая части зеленой обочины.

выложить части кода в интернет. Также будет выложена в [18] dll, которая реализует все перечисленные модули для общего анализа изображений. Программный комплекс, который позволяет строить ростки левых и правых глобальных объектов для любого цветного изображения выложен на сайт [18]. Пример левого ростка глобального множества для изображения рис. 2 приведен на рис. 4.

Подобные определения вводятся для поиска малых и тонких объектов. Единственная разница состоит в том, что в цепочках цветовых сгустков участвуют контрастные доминируемые сгустки. На этом языке могут быть переформулированы многие классические задачи понимания изображений.

Например, так можно сформулировать задачу детектирования и понимания дорожной разметки [14]. При этом удается отыскивать как белую, так и цветную разметку, производить нахождение временной цветной разметки при наличии старой белой и анализировать криволинейную разметку также хорошо, как и прямолинейную.

Поисковая решетка SearchLat активно используется при сегментации и решении задачи понимания изображений. С помощью специальных массивов описывается, какие левые и правые ростки проходят через любой цветовой сгусток. Также отмечается тот факт, что таких ростков не имеется. С помощью SearchLat легко строятся отношения соседства для ростков глобальных множеств [12], что при обычных подходах представляет серьезную проблему.

Таким образом, готовится процедура объединения ростков. Также с помощью SearchLat строится система рассуждений, которая позволяет выделять кандидатов на часть дорожной разметки в полосе. Так как кандидаты на части дороги всегда принадлежат SearchLat, то для любого цветового сгустка достаточно проанализировать его связи в яркостно-цветовой и в геометрической областях с соседними базисными сгустками SearchLat, чтобы отнести его к кандидатам на часть разметки в полосе.

Поисковая решетка SearchLat и структура BunchLoc также позволяют выделять мелкие контрастные объекты на изображениях. Используя структуру BunchLoc, для каждого доминируемого цветового сгустка b можно определить все базисные цветовые сгустки из SearchLat, интервалы локализации которых пересекают интервал локализации b.

Определение 2.2. Доминируемый цветовой сгусток b является контрастным цветовым сгустком в *STG*, если он имеет контрастные цветовые характеристики со всеми базисными сгустками из SearchLat, с которыми он имеет пересекающиеся интервалы локализации.

Замечание. Для доминирующих сгустков определение контрастности было дано в [11]. Оно утверждает, что доминирующий сгусток – контрастный, если его яркостно-цветовые характеристики контрастны к яркостно-цветовым характеристикам соседних доминирующих сгустков.

Используя это определение, с помощью формальных процедур, на *STG* находятся образы контрастных малых объектов из изображения, таких как фары автомобилей, сигнальные огни летательных объектов, сигналы светофоров, стоп-сигналы и сигналы поворотов автомобилей, и т.д. Для подтверждения истинности найденных малых объектов используются данные глобальной сегментации. Эти данные подтверждают, что малый контрастный объект находится рядом с большим контрастным объектом (сигнальные огни вертолета) или внутри него (стоп-сигналы автомобиля). Взаимодействие локального и глобального анализа обеспечивает надежность идентификации.

Дальнейшие операции сегментации и построения систем понимания изображений выполняются без обращения к массиву изображения, путем использования только графа цветовых сгустков и левых, правых ростков глобальных объектов. При этом вместо миллионов пикселов изображения используются только несколько тысяч цветовых сгустков. Отметим, что на данной стадии (сборка объектов из частей) может быть использована семантическая информация, так как мы знаем описание геометрии частей и их яркостно-цветовые характеристики. Также очень важны отношения соседства, полученные с помощью SearchLat. Время, затраченное на сегментацию и решение большого числа задач понимания изображений, при, например, разбиении на 60 полос изображений с

разрешением порядка 1600 × 1200, 1280 × 720, менее 100 мс. Основное время тратится на построение графа цветовых сгустков. При таких разрешениях и при последовательной обработке полос обеспечивается быстродействие 8–12 кадров в секунду. Ясно, что при большом числе процессоров и распараллеливании обработки полос изображения, данное время может быть сильно сокращено, почти до нуля, так как время обработки одной полосы и так небольшое.

В работе предлагаются методы этого сокращения и даются оценки полученного сокращения для нескольких типов современных персональных компьютеров с процессорами с несколькими ядрами и несколькими логическими процессорами. Заметим, что даже простейшие реализованные методы распараллеливания приводят к рекордным результатам решения задач сегментации и построения систем понимания изображений на видео высокого разрешения.

3. МНОГОПОТОЧНАЯ АДАПТАЦИЯ МЕТОДА ГЕОМЕТРИЗОВАННЫХ ГИСТОГРАММ И ОПИСАНИЕ ИТОГОВОЙ СИСТЕМЫ

Данный раздел посвящен простейшей параллельной адаптации метода геометризованных гистограмм. В последующих работах планируется рассмотреть другие способы распараллеливания представленного алгоритма обработки изображений.

Стоит отметить, что для достижения хорошего прироста к скорости обработки видеопоследовательностей, было необходимо в дополнение к алгоритмическому описанию, приведенному в предыдущем разделе, провести комплексный анализ программной реализации рассматриваемого метода анализа изображений.

Приведем краткую характеристику приложения, содержащего исходную реализацию метода геометризованных гистограмм.

Данное приложение написано на языке программирования C++ версии 2003, для отображения графического интерфейса и воспроизведения видеопоследовательностей используется пакет Microsoft Foundation Classes (MFC). Разработка ведется в среде Microsoft Visual Studio 2017.

Для обеспечения возможности запуска приложения на других операционных системах было принято решение перейти к использованию классов STL для работы с файловой системой, библиотеки OpenCV – для работы с видеофайлами, а для сборки исполняемых файлов – использовать утилиту CMake.

Для реализации многопоточного исполнения приложения были применены относительно новые возможности стандартной библиотеки C++, появившиеся, начиная с 11 версии, а именно, классы из пространств имен *std::thread*, *std::mutex* и их методы, определенные в заголовочных файлах <*thread*> и <*mutex*>, соответственно.

Было определено, что наибольшее время занимает исполнение функций, отвечающих за этап сегментации изображений видеопоследовательности. Подробно ознакомиться со связанными метриками можно в следующем разделе.

Метод геометризованных гистограмм изначально спроектирован так, что каждая полоса, на которую разделяется изображение, может быть обработана по отдельности. Поэтому было решено попробовать применить многопоточное программирование именно к обработке этих полос. Стоит отметить, что метод не сводится только к обработке каждой полосы. На данном этапе осуществлена параллельная реализация той части процедуры сегментации, которая занимает 95% времени выполнения в однопоточном режиме. Поэтому не удалось добиться ускорения программы в количество раз, линейно зависящее от числа физических ядер процессора.

За хранение полос изображения отвечает отдельный класс CStrip. В однопоточной реализации на этапе сегментации выполняется несколько циклов, однако есть только один, наиболее требовательный к вычислительным ресурсам, назовем его “главным циклом”. Он производит необходимые действия над объектами указанного класса. В свою очередь, внутри “главного цикла” производится вызов нескольких других функций (около четырех), время исполнения которых в среднем варьируется от 0.04 мс на кадр до 45 мс (при обработке изображения с разрешением 640 × 480 пикселов).

Данный цикл был заменен на одновременное исполнение некоторого количества независимых потоков. В ходе экспериментов было выявлено, что для достижения наиболее оптимального прироста скорости обработки изображений, необходимо запускать число потоков, равное числу полос, на которые разрезается исходное изображение (в зависимости от разрешения, этот параметр устанавливается различным, но чаще всего равен 60 или 72). Стоит отметить, что выбор такого количества потоков (большего, чем количество физических ядер центрального процессора) обусловлен необходимостью обеспечения достаточной универсальности для работы программы с видеопоследовательностями различных разрешений на различных компьютерах, в том числе, на бортовом сервере беспилотного автомобиля.

Для обеспечения корректного одновременного доступа к данным изображения (объект класса

```
auto begin = std::chrono::steady_clock::now();
vision.segment(resized_fr, fr_number);//start segmentation
vision.visualize_markings(resized_fr);//visualize markings
//calculate one frame's time for current frame
float secs = std::chrono::duration_cast<std::chrono::milliseconds>
    (std::chrono::steady_clock::now() - begin).count() / 1000.0;
sum += secs;//all time counter increment
```

Рис. 5. Метод расчета времени выполнения функций с помощью *std::chrono*.

cv::Mat библиотеки OpenCV) используются блокировки с помощью *std::mutex*.

Стоит отметить, что даже на ноутбуке средней ценовой категории 2017 года, имеющем процессор Intel Core i7 с четырьмя физическими ядрами, время обработки одного кадра видеопоследовательности, получаемой в реальном времени с камеры высокого разрешения, уменьшилось более чем в 3 раза и составило менее 30 мс.

4. ПРИМЕНЕНИЯ, РЕЗУЛЬТАТЫ И ОЦЕНКИ

Для оценки скорости работы программы применяются возможности стандартной библиотеки C++, реализованные в классах из пространства имен *std::chrono*. Способ получения временного значения в миллисекундах представлен на рис. 5.

В ходе тестов исходной версии приложения производился замер времени выполнения всей процедуры сегментации в целом, а также более мелких функций, вызываемых из “главного цикла”. Также производилось сравнение с результатами, полученными при запуске многопоточной версии.

Для большей наглядности на рис. 6 приведена схема работы алгоритма, а также выделены те его части, которые теперь выполняются в многопоточном режиме.

Данные собирались при обработке трех качественно разных видеорядов на трех ноутбуках с процессорами уровня Intel Core i5/i7 (седьмого и восьмого поколений), обладающими четырьмя физическими ядрами. Затем производилось усреднение данных. В приведенных далее графиках и таблицах время обработки кадра является вычисленным средним арифметическим по 1000 кадрам из каждого видео.

Считывание и предобработка исходной видеопоследовательности, в частности, уменьшение разрешения, производилось с помощью библиотеки OpenCV.

Стоит учитывать тот факт, что временные затраты на работу функций библиотеки OpenCV (включая считывание видеопотока, изменение разрешения изображения и различную визуализацию на кадре, например, отрисовку найденной методом геометризованных гистограмм дорожной разметки) также малы и составляют около 5–10 мс на один кадр.

Представленная на рис. 7 таблица 1 отражает полученную экспериментальную информацию о времени (в миллисекундах), необходимом на выполнение сегментации одного кадра разными версиями программы (столбец *multithreading state*), реализующей метод геометризованных гистограмм. Столбец *num of strips* отражает количество полос, на которые разрезается изображение, и соответствует количеству итераций “главного цикла” в исходной реализации метода и числу запускаемых потоков в многопоточной версии, соответственно.

На рис. 8 для наглядности представлены графики, содержащие аналогичные данные: по вер-

Таблица 1.

Multithreading state	Num of strips	640 × 480	1280 × 720	1920 × 1080
Off	48	25.381	59.1323	118.512
On	48	10.2064	18.8395	36.169
Off	72	29.9551	66.3633	124.652
On	72	16.3946	22.6448	38.6855
Off	108	35.3268	76.5239	136.415
On	108	24.1019	29.0153	41.0169

Рис. 6. Блок-схема этапов работы (блоков программного комплекса) метода геометризованных гистограмм, серым фоном выделены параллельно исполняемые части метода.

Multithreading state	Num of strips	640×480	1280×720	1920×1080
Off	48	25.381	59.1323	118.512
On	48	10.2064	18.8395	36.169
Off	72	29.9551	66.3633	124.652
On	72	16.3946	22.6448	38.6855
Off	108	35.3268	76.5239	136.415
On	108	24.1019	29.0153	41.0169

Рис. 7. Сравнение временных затрат на сегментацию одного кадра однопоточной и многопоточной реализациями метода геометризованных гистограмм.

тикальной оси – время на его обработку в миллисекундах, по горизонтальной – разрешение обрабатываемого изображения в пикселах.

Программный комплекс, реализованный на основе предложенного подхода, имеет много применений.

На учебном роботе Амуре было реализовано решение задачи поиска и распознавания движущейся трехцветной метки. Робот Амур, управляемый в реальном времени, с обратной связью по зрению, осуществлял движение за человеком, несущим в руке табличку с меткой. При этом человек стремился всячески запутать робота, выходя из кадра и пряча метку. Работа робота демонстрировалась на форуме роботов в присутствии в кадре других людей и различных плакатов на стенах с похожими цветными частями. Робот был запущен десятки раз и всегда осуществлял безошибочное слежение. Запись с камеры робота при движении может быть найдена на сайтах [16, 17].

На основе данного подхода реализуется подсистема компьютерного зрения системы управления беспилотного автомобиля “АвтоНива” [15, 18].

Для работы программы с видеопотоком реального времени в условиях задач, связанных с роботомобилем Нивой, где камеры, применяемые в экспериментах, снимают с частотой 30 кадров в секунду, был выбран оптимальный режим работы программы, а именно, с разбиением кадра на 72 полосы и разрешением 1280 × 720, при котором время обработки составляет в среднем 22.6 миллисекунды на одно изображение. Именно этот режим в условиях ограниченных вычислительных мощностей дает наиболее детализированное распознавание кадра.

В настоящее время разработаны и опробованы на видеопоследовательностях дорожных сцен блоки поиска обочины, неба, дорожной разметки. Результаты обработки могут быть найдены на сайтах [16, 17]. Разработаны алгоритмы и отлаживаются программы поиска и распознавания дорожных знаков (в первую очередь, знаков, связанных с распознаванием выезда на перекрестки), поиска и распознавания сигналов светофоров, нахождения других участников движения в собственной и соседних полосах движения. Также разработана программа поиска сигнальных огней вертолета, и она успешно опробована на имеющихся видеозаписях. Это подтверждает способность метода находить очень маленькие окрашенные объекты на фоне объектов с похожими яркостно-цветовыми характеристиками (небо, растительность). Также применение методов глобального анализа и полная сегментация изображения подтверждают принадлежность сигнальной лампы большому объекту (вертолету, дрону, etc.). Сложный состав образа лампы (крайняя часть имеет красный цвет, а

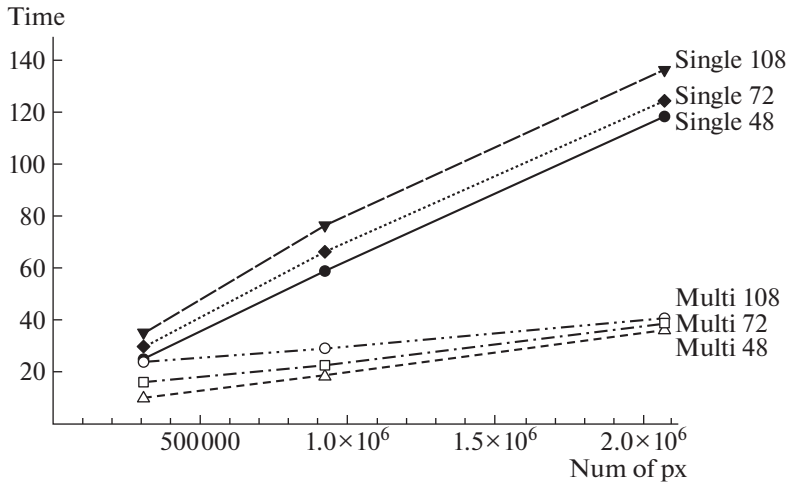


Рис. 8. Сравнение времени обработки одного кадра в однопоточной и многопоточной версиях приложения.

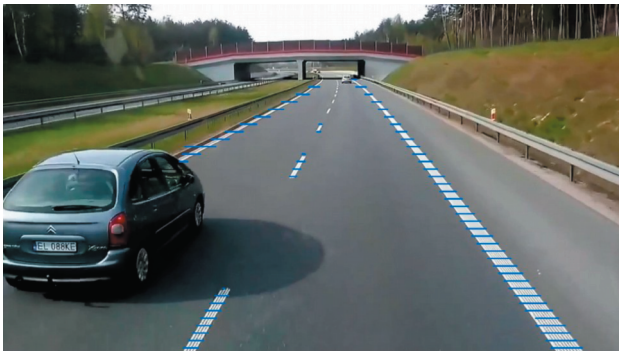


Рис. 9. Пример кадра с выделенной дорожной разметкой.

внутренняя – оранжевый) также удается фиксировать при анализе, что позволяет различить объект на фоне помех. Необходимо также отметить, что разработанные методы нахождения дорожной разметки могут быть применены для нахождения разметок взлетно-посадочной полосы для обеспечения автоматической посадки летательных объектов.

Рисунки 9, 10 показывают результаты обработки кадров видеопоследовательностей для поиска дорожной разметки.

Демонстрируемые примеры показывают работу системы детектирования и понимания разметки в случае заслонения части разметки другим объектом, и когда освещение сильно неоднородно, например, при заезде автомобиля под мост. Существенно, что система правильно находит разметку вне зависимости от интенсивности разметки и ее окружения. Также система демонстрирует хорошие результаты для частично стертой осенне-весенней разметки. Результаты обработки видеопоследовательностей с целью поиска разметки и других объектов на дороге будут размещаться на сайтах [17, 18].

Рис. 10. Пример распознавания дорожной разметки в разных условиях освещенности полотна дороги.

5. ЗАКЛЮЧЕНИЕ

В работе описана многопоточная программная система для сегментации и понимания изображений и кадров видеопоследовательностей в реальном времени. Описаны основные методы и алгоритмы системы. Приведены оценки времени работы системы при решении комплекса задач понимания дорожных сцен. Демонстрируются примеры работы подсистемы поиска разметки на дороге.

Адаптация метода геометризованных гистограмм для работы в многопоточном режиме позволила ускорить обработку одного кадра видеопоследовательности в среднем более чем в 3 раза и добиться хорошего показателя – менее 30 мс на кадр. Такое время обработки можно считать достаточно малым даже при анализе получаемой в реальном времени видеопоследовательности, поскольку большинство современных камер делают 15–60 кадров в секунду.

В дальнейшем планируется применить технологии параллельного программирования на более глубоких уровнях обработки изображений, а также провести замеры времени выполнения программы на более мощном оборудовании, обладающем большим количеством физических ядер процессора.

СПИСОК ЛИТЕРАТУРЫ

1. *Felzenszwalb P.F., Huttenlocher D.P.* Efficient graph-based image segmentation // International Journal of Computer Vision. 2004. V. 59. № 2. P. 167–181.
2. *Shotton J., Winn J., Rother C., Criminisi C.A.* TexstonBoost for image understanding multi-class objects recognition and segmentation by jointly modeling texture, layout, and context // International Journal of Computer Vision. 2004. V. 81. № 1. P. 2–23.
3. *Badrinarayanan V., Kendall A., Cipola R.* SegNet: a deep convolutional encoder-decoder architecture for image segmentation // IEEE Trans. PAMI. 2017. V. 39. № 12. P. 2481–2495.
4. *Tian Y., Gelernter J., Wang X.* Lane marking detection via deep convolutional neural network // Neurocomputing. 2018. V. 280. P. 46–55.
5. *Lin G., Santoso P.S., Lin C., Tsai C., Guo J.* One Stage Detection Network with an Auxiliary Classifier for Real-Time Road Marks Detection // 2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), Honolulu, HI, USA. 2018. P. 1379–1382.
6. *Norote S.P., Bhujbal P.N., Norote A.S., Dhane D.M.* A review of recent advances in lane detection and departure warning system // Pattern Recognition. 2018. V. 73. № 2. P. 216–234.
7. *Son J., Yoo H., Kim S., Sohn K.* Real-time invariant lane detection for lane departure warning system Expert Systems with Applications. 2015. V. 42. P. 1816–24.
8. *Chen L., Li J., Zhou J., Jiang M.* “Multithreading Method to Perform the Parallel Image Registration”, 2009 International Conference on Computational In-

telligence and Software Engineering, Wuhan. 2009. P. 1–4.
9. *Saaidon N., Sediono W.* "Multicolour object detection using multithreading", 2015 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE), Langkawi. 2015. P. 48–52.
10. *Kiy K.I.* A new real-time method for description and generalized segmentation of color images // Pattern Recognition and Image Analysis. 2010. V. 20. № 2. P. 398–401.
11. *Kiy K.I.* Segmentation and detection of contrast objects and their application in robot navigation // Pattern Recognition and Image Analysis. 2015. V. 25. № 2. P. 338–346.
12. *Kiy K.I.* A new method of global image analysis and its application in understanding road scenes // Pattern Recognition and Image Analysis. 2018. V. 28. № 3. P. 483–494.
13. *Kiy K.I.* An image understanding system based on the geometrized histograms method: finding the sky in road scenes // CEUR Workshop Proceedings. V. 2210. P. 291–299. http://ceur-ws.org/Vol-2210/paper38.pdf/.
14. *Dosaev R.V., Kiy K.I.* A new real-time method for finding temporary and permanent road marking // CEUR Workshop Proceedings. V. 2391. P. 86–96. http://ceur-ws.org/Vol-2391/paper12.pdf/
15. *Подопросветов А.В., Павловский В.Е., Кий К.И., Анохин Д.А.* Робоавтомобиль Нива: алгоритмы управления. Сборник трудов ПУМСС XXI Международной конференции. 2019. Т. II. С. 102–106.
16. http://video.mail.ru/mail/kikip_46/_myvideo/
17. https://www.facebook.com/100004887018729/videos
18. http://project1054516.tilda.ws/

ПРОГРАММИРОВАНИЕ, 2020, № 6, с. 55–66

ПАРАЛЛЕЛЬНОЕ И РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ

УДК 004.724.4

ОТОБРАЖЕНИЕ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ НА РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ, ИСПОЛЬЗУЮЩИЕ ТЕХНОЛОГИЮ RapidIO

© 2020 г. Н. И. Вьюкова[a,*], В. А. Галатенко[a,], А. Н. Павлов[a,***], С. В. Самборский[a,****]**

[a] Федеральное государственное учреждение "Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук", Москва, Нахимовский проспект, д. 36, к. 1, 117218 Россия

**E-mail: qniva@yandex.ru*

***E-mail: galat@niisi.ras.ru*

****E-mail: antony@niisi.msk.ru*

*****E-mail: sambor@niisi.ras.ru*

Поступила в редакцию 18.05.2020 г.
После доработки 15.06.2020 г.
Принята к публикации 17.06.2020 г.

Статья посвящена вопросам отображения параллельных вычислений на распределенные системы с коммутационной средой RapidIO. Распределенная система представляет собой множество вычислительных узлов с заданными величинами производительности и узлов-коммутаторов, связанных сетевыми соединениями с заданными величинами пропускной способности. Под параллельным вычислением понимается множество процессов, взаимодействующих посредством передачи потоков данных, где для каждого потока требуется определенная пропускная способность сетевого маршрута, а для каждого процесса – определенная производительность. Требуется назначить вычислительные узлы для процессов и определить маршруты для потоков данных, а также построить таблицы маршрутизации для коммутаторов так, чтобы обеспечить заданные пропускные способности для потоков данных и заданные величины производительности для процессов. Данная задача в общем случае является NP-полной. В работе предложен подход, основанный на методах целочисленного линейного программирования (ЦЛП), позволяющий корректно отобразить параллельное вычисление на распределенную систему и оптимизировать отображение по ряду характеристик.

DOI: 10.31857/S0132347420060084

1. ВВЕДЕНИЕ

Процессоры семейства Комдив64, разработанные в ФГУ ФНЦ НИИСИ РАН, предназначены для широкого спектра высокопроизводительных вычислительных приложений, в том числе, приложений для управления различными техническими устройствами в реальном времени.

Для эффективной работы подобных приложений может требоваться большой объем вычислительных ресурсов. Одним из общепринятых способов реализации высокопроизводительных вычислений в настоящее время является применение распределенных вычислительных систем. Важными преимуществами этого подхода являются масштабируемость и гибкость, поскольку вычислительный комплекс может быть спроектирован в соответствии с потребностями приложения.

В ФГУ ФНЦ НИИСИ РАН разработаны процессорные и мезонинные модули на базе микропроцессоров КОМДИВ64-РИО и КОМДИВ128-РИО, а также коммутаторы и другие аппаратные средства для построения многопроцессорных комплексов, связанных посредством коммуникационной сети RapidIO [1].

RapidIO – это высокопроизводительный интерфейс передачи данных для соединения микросхем на одной печатной плате, а также для соединения между собой нескольких печатных плат. Данный интерфейс был разработан для применения во встраиваемых системах. RapidIO оптимизирован для энергоэффективных связей процессор-процессор с минимальными задержками в устойчивых к ошибкам системах, узлы которых удалены друг от друга на расстояние не более ки-

лометра. Согласно [2], по таким характеристикам как пропускная способность, задержки при передаче пакетов, надежность, масштабируемость, RapidIO превосходит другие коммуникационные средства.

В [3] представлена технология разработки систем цифровой обработки сигналов для многопроцессорных комплексов на базе процессоров архитектуры Комдив и коммуникационной среды RapidIO (Комдив-RapidIO). В [4] рассмотрен опыт реализации алгоритма геометрического многосеточного метода из пакета NAS Parallel Benchmarks на распределенном комплексе Комдив-RapidIO. Поддержка коммуникационной сети RapidIO на уровне операционной системы реального времени (ОСРВ) Багет представлена в [5, 6]. На прикладном уровне создание программного обеспечения для комплексов Комдив-RapidIO обеспечивается библиотекой параллельной обработки сигналов (БПОС) [7]. Реализован также ряд инструментальных средств для конфигурирования и загрузки параллельных программ на распределенные комплексы Комдив-RapidIO [8], а также для интерактивной ручной оптимизации привязки параллельно выполняемых процессов к вычислительным узлам целевого многопроцессорного комплекса [9].

Тем не менее, вопрос оптимального отображения параллельного вычисления на распределенную вычислительную систему остается нерешенным. Процессы взаимодействуют между собой путем передачи сообщений. Для эффективного выполнения параллельного вычисления в целом необходимо, чтобы сетевые маршруты обеспечивали определенный темп передачи данных между взаимодействующими процессами. Этого можно добиваться как путем варьирования назначения вычислительных узлов для процессов, так и путем изменения схемы маршрутизации в сети RapidIO. При этом вычислительный узел, назначаемый для каждого процесса, должен обладать определенными свойствами, такими как уровень производительности или наличие некоторого сопроцессора.

Для небольшого числа процессов оптимальное (или удовлетворительное) отображение и подходящая схема маршрутизации могут быть подобраны вручную экспериментальным путем. Но когда число процессов исчисляется десятками, такой экспериментальный подбор оказывается весьма трудоемким и не гарантирует оптимального результата.

Статья посвящена вопросам точного решения задачи об отображении заданного параллельного вычисления на распределенную вычислительную систему с сетью произвольной топологии и статической маршрутизацией сетевой среды. Данная задача является NP-полной, и в настоящей работе для ее решения предложен подход, использующий методы целочисленного линейного программирования (ЦЛП).

Последующая часть статьи построена по следующему плану. В разделе 2 обсуждаются существующие работы в данной области. В разделе 3 рассматриваются формальные модели параллельного вычисления и распределенной вычислительной системы на базе коммуникационной среды RapidIO. В разделе 4 представлены формулировки двух ЦЛП-задач – задачи нахождения схемы маршрутизации в условиях, когда процессам уже назначены узлы распределенной системы, и более сложной задачи, включающей назначение вычислительных узлов для процессов и нахождение схемы маршрутизации. В разделе 5 приведены примеры решения этих двух ЦЛП-задач. В заключение анализируются результаты проведенных исследований и рассматриваются направления дальнейших исследований.

2. ДРУГИЕ РАБОТЫ В ЭТОЙ ОБЛАСТИ

В сфере высокопроизводительных вычислений требования к вычислительным мощностям постоянно возрастают. В настоящее время практически единственный способ удовлетворить эти растущие требования заключается в использовании параллельных вычислительных конфигураций, в частности, распределенных многопроцессорных систем. В связи с этим большое значение приобретает развитие методов, позволяющих автоматически распределить параллельно выполняемые процессы по узлам компьютерной сети с соблюдением тех или иных требований. Исследования в данной области ведутся уже на протяжении не одного десятилетия. Однако их актуальность остается высокой в силу постоянного развития архитектур распределенных систем и сетевых технологий, а также из-за разнообразия требований, которые могут предъявляться к искомым отображениям.

Многие варианты задачи отображения параллельных вычислений на распределенные системы являются NP-полными, поэтому для ее решения преимущественно используются эвристические подходы. Обзор подобных подходов можно найти в [10]. В связи с настоящим исследованием наибольший интерес представляют работы, посвященные точным методам решения данной задачи с использованием таких формализмов как ЦЛП, смешанное целочисленное линейное программи-

рование (Mixed Integer Linear Programming, MILP), задача выполнимости формул в теориях (Satisfiability Modulo Theories, SMT). Применение точных методов оправдано, если отображение параллельной задачи на многопроцессорную конфигурацию определяется в процессе ее разработки и отладки производительности. Точные методы могут применяться для оценки эффективности эвристических алгоритмов, как в [11], [12]. Они также могут применяться в сочетании с эвристическими подходами, как в [13].

Формулировки задач отображения параллельных вычислений на распределенную вычислительную систему могут существенно отличаться, во-первых, из-за различий архитектур используемых вычислительных систем и коммуникационных моделей, во-вторых, из-за различий в критериях допустимости и оптимальности искомых отображений.

Значительное число работ в данной области посвящено многопроцессорным системам на кристалле (Multiprocessor system-on-chip, MPSoC) с сетью, имеющей структуру 2D-решетки. Например, в [14] рассматривается выполнение параллельных вычислений на системе MPSoC и ставится задача найти отображение множества процессов на множество процессоров и определить приоритеты коммуникаций между процессами так, чтобы минимизировать общее время выполнения. В работах [15, 16] рассматриваются вопросы минимизации коммуникационных издержек между взаимодействующими процессами, выполняющимися в многопроцессорных конфигурациях, за счет дублирования процессов на нескольких вычислительных узлах.

Целый ряд работ посвящен актуальным темам отображения параллельных вычислений на многопроцессорные вычислительные системы с оптимизацией энергопотребления [13, 17, 18] или тепловыделения [11].

В ряде случаев при поиске отображений параллельных вычислений на распределенную систему приходится рассматривать многокритериальную оптимизацию. Например, в [19] отображения параллельных вычислений на узлы MPSoC оптимизируется как по сбалансированности загрузки узлов, так и по коммуникационным издержкам. Обычно в таких случаях применяют целевую функцию, являющуюся линейной комбинацией двух (или более) параметров. В [19] предлагается подход, позволяющий находить “неулучшаемые” решения из множества Парето.

В большинстве работ формулировки ЦЛП или MILP-задач включают спецификацию расписания для параллельных вычислений, с тем чтобы удовлетворить заданные временные ограничения для систем реального времени, как в [11, 13], либо минимизировать общее время выполнения, см. [14, 16].

В данной работе временные характеристики выполнения параллельных вычислений не рассматриваются. Основной ее вклад состоит в том, что, помимо отображения процессов на вычислительные узлы распределенной системы, в результате решения ЦЛП-задачи вычисляется также схема маршрутизации, которая гарантирует требуемую пропускную способность маршрутов между вычислительными узлами, на которых выполняются взаимодействующие процессы. В других работах либо подразумевается детерминированный алгоритм маршрутизации, например, XY-маршрутизация в системах MPSoC, (см. [11, 16]), либо предполагается, что таблица маршрутизации предопределена, как в [20].

В [20] представлена постановка задачи, наиболее близкая к нашей. В этой работе используется формализм SMT для поиска отображения параллельно выполняемых процессов на вычислительные узлы сети произвольной топологии с целью минимизации общего сетевого трафика и обеспечения сбалансированной загрузки вычислительных узлов. Как и в нашей работе, в [20] учитываются ограничения по пропускной способности каждого сетевого соединения: суммарная потребность всех маршрутов, использующих некоторое соединение, не должна превышать пропускной способности этого соединения. Однако в [20], в отличие от нашей работы, таблица маршрутизации считается предопределенной, и варьируется только назначение вычислительных узлов для процессов.

3. ФОРМАЛЬНЫЕ МОДЕЛИ

3.1. Модель распределенной вычислительной системы

Коммуникационная среда RapidIO представляет собой сеть с коммутацией пакетов, состоящую из узлов, соединенных физическими каналами связи [1]. Структура распределенных систем на базе аппаратуры линейки Комдив под управлением ОСРВ Багет, подробно описана в [5]. Формальная модель такой распределенной системы представлена в [21]. В данной работе, как и в [6], мы будем рассматривать упрощенную модель, включающую узлы двух типов:

– вычислительный узел – оконечное устройство, имеющее числовой идентификатор и, как правило, только один порт;

– коммутатор – устройство, предназначенное для маршрутизации, которое всегда имеет несколько портов.

Коммутатор работает под управлением таблицы маршрутизации, которая записывается во время инициализации коммуникационной среды RapidIO. Есть два типа коммутаторов. Коммутатор первого типа имеет одну общую таблицу маршрутизации для всех портов, в коммутаторе второго типа имеется индивидуальная таблица для каждого порта.

В коммутаторах первого типа выходной порт, с которого будет отправлен полученный пакет данных, однозначно определяется указанным в этом пакете идентификатором вычислительного узла – получателя. Это означает, что если маршруты, по которым передаются данные одному получателю, пересекаются на некотором коммутаторе, то далее они совпадают.

В коммутаторах второго типа выходной порт, с которого будет отправлен полученный пакет данных, определяется идентификатором получателя и номером порта, на который этот пакет пришел. Таким образом, пакеты к одному получателю могут уйти с коммутатора по разным соединениям, если они поступили в него по разным соединениям.

Последовательность соединений, через которые проходит пакет от отправителя до получателя, называется *маршрутом*. Множество маршрутов для всех пар отправителей и получателей, между которыми происходит передача данных, называется *схемой маршрутизации*.

Представим теперь формальную модель распределенной системы, которая будет использована в формулировках ЦЛП-задач в разделе 4.

Распределенная система представляется в виде ориентированного размеченного (нагруженного) графа

$$G = (H, C_1, C_2, L, b, perf),$$

где H – множество вычислительных узлов, C_1 – множество коммутаторов первого типа, C_2 – множество коммутаторов второго типа. Обозначим через *Node* множество вычислительных узлов и коммутаторов: $Node = H \cup C_1 \cup C_2$. Тогда $L \subseteq Node \times Node$ – множество дуг, соответствующих соединениям между узлами распределенной системы. Отображения b: $L \to \mathbb{N}$ и *perf*: $H \to \mathbb{N}$ задают, соответственно, пропускную способность соединений и производительность вычислительных узлов в некоторых условных единицах.

На практике в среде RapidIO соединения двунаправленные, с одинаковой пропускной способностью в обе стороны. Поэтому граф G можно было бы определить как неориентированный. Тем не менее мы рассматриваем его как ориентированный, поскольку в формулировках ЦЛП-задач соединения (n, n') и (n', n) определяются независимо друг от друга.

3.2. Модель параллельного вычисления

Параллельное вычисление мы будем представлять в виде ориентированного размеченного графа

$$PG = (P, S, v, req)$$

где P – множество процессов, реализующих параллельное вычисление, $S \subseteq P \times P$ – множество дуг, связывающих взаимодействующие процессы: $s = (p, p') \in S$, если процесс p отправляет сообщения процессу p'. Допускаются как ациклические графы, так и графы с циклами. Дуги $s \in S$ мы далее будем называть потоками данных. Отображения v: $S \to \mathbb{N}$ и *req*: $P \to \mathbb{N}$ определяют, соответственно, требуемую для каждого потока данных пропускную способность сетевого маршрута и требуемую для каждого процесса производительность вычислительного узла.

4. ФОРМУЛИРОВКИ ЦЛП-ЗАДАЧ

В этом разделе мы рассмотрим две формулировки ЦЛП-задач, связанных с отображением параллельного вычисления на распределенную вычислительную систему.

В первой задаче предполагается, что множество процессов P параллельного вычисления $PG = (P, S, v, req)$ уже отображено на множество вычислительных узлов H распределенной вычислительной системы $G = (H, C_1, C_2, L, b, perf)$. Требуется определить схему маршрутизации, обеспечивающую заданную пропускную способность для потоков данных между процессами, а также вычислить таблицы маршрутизации для коммутаторов.

Формулировка второй ЦЛП-задачи включает подбор вычислительных узлов из H для процессов из P, а также нахождение схемы и таблиц маршрутизации.

В качестве инструмента для описания и решения ЦЛП-задач применялся пакет GLPK [22].

4.1. Задача маршрутизации

Пусть заданы параллельное вычисление $PG = (P, S, v, req)$ и распределенная вычислительная система $G = (H, C_1, C_2, L, b, perf)$. Первая ЦЛП-задача, которую мы рассмотрим, это задача вычис-

ления схемы маршрутизации и таблиц маршрутизации в условиях, когда процессы параллельного вычисления уже назначены на вычислительные узлы этой системы. В этой задаче отображения *perf* и *req*, связанные с производительностью, не рассматриваются.

Пусть определено отображение m: $P \to H$. Тем самым однозначно определяется множество $S' \subseteq \subseteq H \times H$ потоков данных между вычислительными узлами: $s' = (h_1, h_2) \in S'$, если существует поток $s = (p_1, p_2) \in S$, такой что $m(p_1) = h_1$, $m(p_2) = h_2$. Будем называть узел h_1 отправителем, а h_2 получателем потока данных s'. Требуемая пропускная способность для потока s', $v'(s')$, полагается равной сумме $v(s)$ по всем $s = (p_1, p_2) \in S$, таким что $m(p_1) = h_1$, $m(p_2) = h_2$.

Константы, переменные, отображения и подмножества, используемые в формулировках 1-й и 2-й ЦЛП-задач.

Константный массив LN определяет, как связаны соединения $l \in L$ и вершины $n \in Node$ графа G, LN: $L \times Node \to \{-1, 0, 1\}$. Пусть $l = (n_1, n_2)$ – сетевое соединение, направленное от узла n_1 к узлу n_2.

$$LN[l,n] = \begin{cases} -1 & iff \quad n = n_1 \\ 1 & iff \quad n = n_2 \\ 0 & iff \quad n \neq n_1, \quad n \neq n_2 \end{cases}$$

Аналогично введем константный массив SN, определяющий связь между потоками данных и узлами вычислительной системы: SN: $S' \times Node \to$ $\to \{-1, 0, 1\}$. Пусть $s = (n_1, n_2)$ – поток данных от узла n_1 к узлу n_2.

$$SN[s,n] = \begin{cases} -1 & iff \quad n = n_1 \\ 1 & iff \quad n = n_2 \\ 0 & iff \quad n \neq n_1, \quad n \neq n_2 \end{cases}$$

Значение 0 показывает, что узел n не является ни начальным, ни конечным для потока s (хотя может являться промежуточным узлом в маршруте для потока s). Этот массив используется только в формулировке первой ЦЛП-задачи.

Введем также обозначения для следующих подмножеств соединений: $Lout_1$ – множество соединений, исходящих из коммутаторов первого типа; $Lout_2$ – множество соединений, исходящих из коммутаторов второго типа; Lin_2 – множество соединений, входящих в коммутаторы второго типа.

Отношение следования, *next*, на множестве соединений используется в формулировках ограничений, обеспечивающих связность маршрутов: *next*(l, l'), если соединение l входит в коммутатор, из которого исходит l'.

Основная переменная, описывающая искомые маршруты для потоков данных из S', это переменная R, которая имеет тип массива бинарных значений: R: $L \times S' \to \{0, 1\}$. Для $l \in L$, $s \in S'$ значение $R[l, s] = 1$, если соединение l используется в маршруте для потока данных s; в противном случае $R[l, s] = 0$.

Массив переменных TC_1 определяет таблицы маршрутизации для коммутаторов первого типа, TC_1: $Lout_1 \times H \to \{0, 1\}$. $TC_1[l, h] = 1$, если соединение $l \in Lout_1$ используется для передачи сообщений, адресованных вычислительному узлу $h \in H$.

Массив переменных TC_2 определяет таблицы маршрутизации для коммутаторов второго типа, TC_2: $Lin_2 \times Lout_2 \times H \to \{0, 1\}$. Для $h \in H$, $l \in Lin_2$, $l' \in Lout_2$, $TC_2[l, l', h] = 1$, если выходное соединение l' используется для дальнейшей пересылки сообщений узлу h, пришедших по входному соединению l. Это имеет смысл, если l и l' связаны с одним коммутатором; переменные для l и l', не связанных с одним коммутатором, не создаются.

В определениях переменных TC_1 и TC_2 сам коммутатор, для которого заполняется таблица маршрутизации, явно не фигурирует. Но для каждой переменной этих массивов коммутатор однозначно определяется по соединению (l для TC_1, l или l' для TC_2).

Маршруты и таблицы коммутации должны удовлетворять ряду ограничений. Это ограничения, связанные со спецификой коммутационной среды RapidIO, ограничения, обеспечивающие заданную пропускную способность для потоков данных между процессами, ограничения, исключающие маршруты с петлями.

Ограничение на маршруты для потоков данных. Следующее ограничение указывает, что каждый маршрут возникает в узле-отправителе соответствующего потока данных и заканчивается в узле-получателе, а в остальные узлы из *Node* входит столько же раз, сколько выходит (не более одного, если маршрут не имеет петель):

$$\forall s \in S', \quad \forall n \in Node$$
$$\sum_{l \in L} LN[l,n] \cdot R[l,s] = SN[s,n] \qquad (1)$$

Ограничения по пропускным способностям. Сумма требуемых пропускных способностей потоков данных, чьи маршруты проходят через некоторое соединение, не должна превышать пропускную способность соединения:

$$\forall l \in L \quad \sum_{s \in S'} v'(s) \cdot R[l,s] \le b(l) \tag{2}$$

Ограничения для вычислительных узлов. Вычислительный узел не может быть транзитным, то есть ни в одном маршруте не может быть использовано более одного соединения, связанного с таким узлом:

$$\forall h \in H, \quad \forall s \in S' \quad \sum_{l \in L: LN[l,h] \ne 0} R[l,s] \le 1 \tag{3}$$

Ограничения на вход и выход из коммутаторов. Маршрут не может использовать два или более соединений, входящих в один и тот же коммутатор, а также два или более соединений, исходящих из одного и того же коммутатора:

$$\forall c \in C_1 \cup C_2, \quad \forall s \in S'$$

$$\sum_{l \in L: LN[l,c]=1} R[l,s] \le 1 \tag{4}$$

$$\forall c \in C_1 \cup C_2, \quad \forall s \in S'$$

$$\sum_{l \in L: LN[l,c]=-1} R[l,s] \le 1 \tag{5}$$

Эти ограничения исключают маршруты с петлями, проходящие через один и тот же коммутатор второго типа более одного раза. Наличие таких маршрутов не является ошибкой для коммутаторов типа 2, но мы исключаем их из соображений эффективности передачи данных.

Ограничения для таблиц маршрутизации коммутаторов первого типа. Следующее ограничение отражает специфику коммутаторов первого типа:

$$\forall h \in H, \quad \forall c \in C_1$$

$$\sum_{l \in Lout_1: LN[l,c]=-1} TC_1[l,h] \le 1 \tag{6}$$

Оно указывает, что для каждого коммутатора $c \in C_1$ и для каждого вычислительного узла h, не более одного соединения, исходящего из c, предназначено для пересылки сообщений узлу h.

Следующее ограничение указывает, что соединение l, исходящее из коммутатора первого типа, может быть использовано в маршруте для потока передачи данных s, ведущего в вычислительный узел h, только если l "предназначено" для пересылок в узел h.

$$\forall l \in Lout_1, \quad \forall s = (h', h) \in S'$$

$$R[l,s] \le TC_1[l,h] \tag{7}$$

Ограничения для таблиц маршрутизации коммутаторов второго типа. Следующее ограничение описывает специфику коммутаторов второго типа:

$$\forall h \in H, \quad \forall c \in C_2, \quad \forall l \in Lin_2 : LN[l,c] = 1$$

$$\sum_{l' \in Lout_2: LN[l',c]=-1} TC_2[l,l',h] \le 1 \tag{8}$$

Для каждого вычислительного узла h, для каждого коммутатора c второго типа и для каждого соединения l, входного для c, не более одного выходного соединения из этого же коммутатора может быть предназначено для пересылки сообщений, пришедших по входному соединению l и адресованных узлу h.

Наконец, определим ограничение на использование соединений из Lin_2, $Lout_2$ в маршрутах для потоков данных.

$$\forall s = (h', h) \in S' \quad \forall l \in Lin_2, \quad \forall l' \in Lout_2 : next(l,l')$$

$$R[l,s] + R[l',s] \le TC_2[l,l',h] + 1 \tag{9}$$

Переменные R и TC_2 принимают значения 0, 1, то есть их можно рассматривать как булевы переменные. Поэтому неравенство (9) эквивалентно логическому высказыванию

$$R[l,s] \;\&\; R[l',s] \Rightarrow TC_2[l,l',h]$$

Таким образом, два соединения l, l', из которых первое входит в коммутатор, а второе исходит из него, могут быть использованы в маршруте для потока данных, только если l' "предназначено" для пересылок получателю этого потока, пришедших по соединению l.

Целевая функция. Представленные выше ограничения гарантируют подбор маршрутов, обеспечивающих заданные пропускные способности для всех потоков данных. С помощью целевой функции можно обеспечить оптимизацию решения по различным параметрам. В данной реализации минимизируется линейная комбинация следующих переменных: максимальная длина маршрута ($R_{\max}$), суммарная длина маршрутов (R_{total}), суммарное число записей во всех таблицах маршрутизации (TC_{total}).

Для переменной $R_{\max}$ определены следующие ограничения:

$$\forall s \in S' \quad \sum_{l \in L} R[l,s] \le R_{\max} \tag{10}$$

Переменная R_{total} определяется как сумма элементов массива R, а переменная TC_{total} – как сум-

ма элементов TC_1 и TC_2. Минимизируемая целевая функция имеет следующий вид:

$$1000 \cdot R_{\max} + 10 \cdot R_{\text{total}} + TC_{\text{total}} \qquad (11)$$

Отметим, что ограничения (1)–(9) не запрещают несвязные маршруты с циклами, отдельными от основного маршрута. Такие маршруты исключаются только оптимизацией по R_{total} в целевой функции.

Несложно модифицировать целевую функцию, чтобы оптимизировать решение и по некоторым другим параметрам, которые обсуждаются в заключении.

4.2. Задача подбора вычислительных узлов и схемы маршрутизации

Пусть заданы параллельное вычисление $PG = (P, S, v, req)$ и распределенная вычислительная система $G = (H, C_1, C_2, L, b, perf)$. Формулировка второй ЦЛП-задачи отличается от представленной выше тем, что включает подбор вычислительных узлов из H для процессов из P. Допускается отображение нескольких процессов на один узел, имеющий достаточную производительность. Но отображение одного процесса на несколько узлов невозможно. Если возникает такая необходимость, например, когда требуемая для процесса производительность превышает производительность имеющихся вычислительных узлов, такой процесс должен быть заранее распараллелен на несколько процессов.

В этой формулировке будут использованы константы, переменные и другие обозначения, введенные в п. 4.1, а также несколько дополнительных констант и одна переменная.

Константы и переменные, используемые в формулировке 2-й ЦЛП-задачи

В предыдущей формулировке мы использовали константный массив SN, определяющий связь между потоками данных и узлами вычислительной системы. В данной формулировке мы введем вместо этого константный массив: SP: $S \times P \to \{-1, 0, 1\}$, описывающий связь между потоками данных и процессами параллельного вычисления. Пусть $s = (p_1, p_2)$ – поток данных от процесса p_1 к процессу p_2.

$$SP[s, p] = \begin{cases} -1 & iff \quad p = p_1 \\ 1 & iff \quad p = p_2 \\ 0 & iff \quad p \neq p_1, \quad p \neq p_2 \end{cases}$$

Искомое отображение множества процессов P на множество вычислительных узлов H представим в виде переменной M, имеющей тип массива бинарных значений: M: $H \times P \to \{0, 1\}$. Для $h \in H$, $p \in P$ значение $M[h, p] = 1$, если процессу p назначен вычислительный узел h.

Ограничения на отображение M. С отображением M связано два очевидных ограничения. Первое – каждому процессу должен быть назначен в точности один вычислительный узел:

$$\forall p \in P \quad \sum_{h \in H} M[h, p] = 1 \qquad (12)$$

Второе – сумма потребностей в производительности процессов, назначенных на один вычислительный узел, не должна превышать производительности этого узла:

$$\forall h \in H$$
$$\sum_{p \in P} req(p) \cdot M[h, p] \leq perf(h) \qquad (13)$$

Ограничение на маршруты для потоков данных. В п. 4.1 мы ввели ограничение (1), указывающее, что каждый маршрут возникает в узле-отправителе и заканчивается в узле-получателе, а в остальные узлы из *Node* входит столько же раз, сколько выходит. Теперь должно быть учтено отображение процессов на вычислительные узлы. Для вычислительных узлов это ограничение принимает следующий вид:

$$\forall s \in S, \quad \forall h \in H$$
$$\sum_{l \in L} LN[l, h] \cdot R[l, s] = \sum_{p \in P} SP[s, p] \cdot M[h, p] \qquad (14)$$

В коммутаторе поток заведомо не начинается и не заканчивается; для коммутаторов это ограничение выглядит следующим образом:

$$\forall s \in S, \quad \forall c \in C_1 \cup C_2$$
$$\sum_{l \in L} LN[l, c] \cdot R[l, s] = 0 \qquad (15)$$

Ограничение (2) по пропускной способности соединений, ограничение (3), запрещающее использовать вычислительные узлы как транзитные, и ограничения (4), (5), запрещающие многократное прохождение маршрута через коммутатор, остаются без существенных изменений. Отличие лишь в том, что вместо множества потоков данных между вычислительными узлами S' используется множество потоков данных между процессами S.

Ограничения для коммутаторов первого и второго типа. Ограничения (6) и (8), отражающие специфику коммутаторов первого и второго типов, остаются без изменений, поскольку они сформулированы только в терминах распределенной системы.

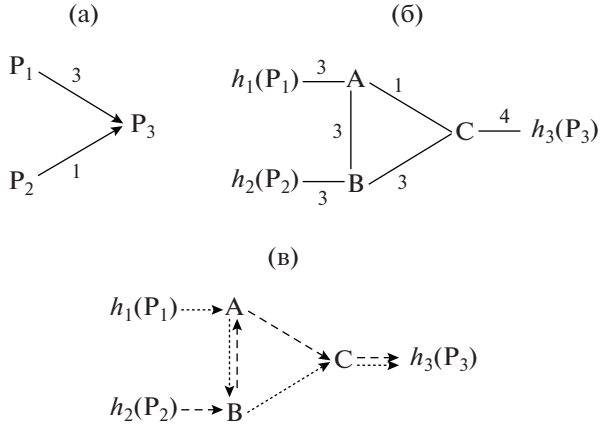


Рис. 1. Пример 1, решение первой ЦЛП-задачи: а) параллельное вычисление; б) распределенная система с заданным отображением процессов на вычислительные узлы; в) решение, маршруты для потоков данных $P_1 \to P_3$ и $P_2 \to P_3$.

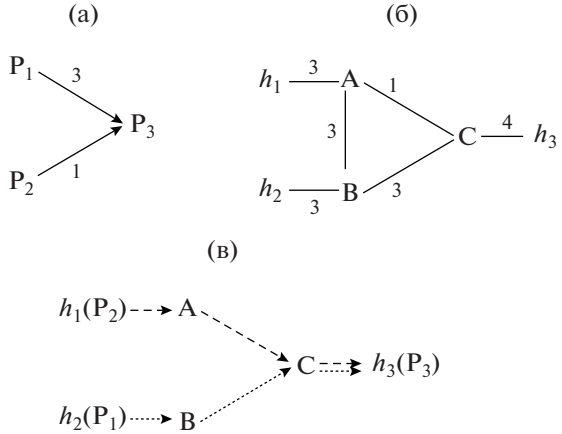


Рис. 2. Пример 1, решение второй ЦЛП-задачи: а) параллельное вычисление; б) распределенная система; в) решение, отображение процессов на вычислительные узлы и маршруты для потоков $P_1 \to P_3$ и $P_2 \to P_3$.

Небольшие отличия возникают в ограничениях (7) и (9) в связи с тем, что в первой формулировке мы рассматривали потоки данных между вычислительными узлами из H, а в данной формулировке потоки данных соединяют процессы из P. Теперь соответствующие ограничения описывают, что в маршруте для потока данных процессу-получателю p можно использовать только выходные соединения, предназначенные таблицей маршрутизации для узла, в который назначен процесс p. Таким образом, ограничение (7) приобретает вид:

$$\forall l \in Lout_1, \quad \forall s = (p', p) \in S, \quad \forall h \in H$$

$$M[h, p] + R[l, s] \le TC_1[l, h] + 1 \qquad (16)$$

А ограничение (9) заменяется на:

$$\forall s = (p', p) \in S,$$
$$\forall l \in Lin_2, \quad \forall l' \in Lout_2 : next[l, l'], \quad \forall h \in H$$

$$M[h, p] + R[l, s] + R[l', s] \le TC_2[l, l', h] + 2 \qquad (17)$$

Переменные, связанные с целевой функцией, ограничение (10), а также целевая функция (11) определяются так же, как в первой задаче.

5. ПРИМЕРЫ

В этом разделе мы рассмотрим примеры решения ЦЛП-задач, представленных выше. Примеры демонстрируют работоспособность предложенных моделей, а также эффект от использования целевой функции. Простой пример, представленный на рис. 1, иллюстрирует специфику двух типов коммутаторов и различие между формулировками двух ЦЛП-задач.

На рис. 1а) показан граф параллельного вычисления из трех процессов P_1, P_2, P_3 с потоками данных $P_1 \to P_3$ и $P_2 \to P_3$ с требованиями по пропускной способности 3 и 1 соответственно.

На рис. 1б) показан граф распределенной системы с тремя вычислительными узлами h_1, h_2, h_3 и отображенными на них процессами P_1, P_2, P_3 соответственно. Распределенная система включает также три коммутатора: A и B – второго типа, C – первого типа. Сплошными линиями показаны пары разнонаправленных соединений между узлами, а цифры показывают пропускные способности соединений. На рис. 1в) показано решение первой ЦЛП-задачи: маршруты, соответствующие потокам данных $P_1 \to P_3$ (пунктирные линии) и $P_2 \to P_3$ (штриховые линии).

Можно видеть, что коммутаторы A и B направляют сообщения адресату h_3 с разных портов в зависимости от того, с какого узла пришло сообщение, поэтому здесь существенно, что оба они относятся ко второму типу. Легко видеть, что это решение единственное, поэтому для случая, когда хотя бы один из коммутаторов A, B относится к первому типу, решения не существует.

Хотя суммарная потребность в пропускной способности для двух потоков данных, проходящих между коммутаторами A и B, равна 4, а на рисунке для (A, B) указана пропускная способность 3, полученное решение корректно, поскольку эти потоки используют два разнонаправленных соединения: (A, B) и (B, A), каждое с пропускной способностью 3.

Рассмотрим теперь решение 2-й ЦЛП-задачи для аналогичного примера, который изображен на рис. 2. Здесь считается, что все вычислительные узлы имеют производительность 1, и потреб-

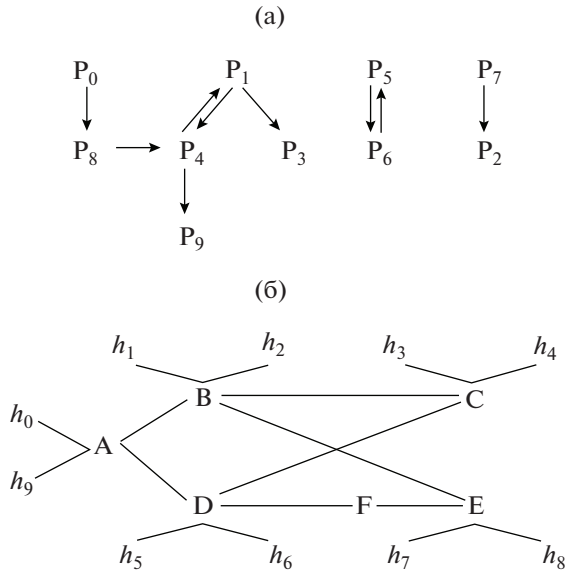


Рис. 3. Пример 2. а) параллельное вычисление; б) распределенная система.

ности всех процессов в производительности также равны 1. На рис. 2а) показан граф параллельного вычисления. На рис. 2б) изображен граф распределенной системы, но без предопределенного отображения процессов на вычислительные узлы. На рис. 2в) показано решение второй ЦЛП-задачи – отображение процессов на вычислительные узлы и маршруты, соответствующие потокам данных $P_1 \to P_3$ и $P_2 \to P_3$.

Назначение вычислительных узлов для процессов, найденное в результате решения второй ЦЛП-задачи, позволило проложить более короткие маршруты для потоков данных $P_1 \to P_3$ и $P_2 \to P_3$, чем в предыдущем случае, без встречных пересылок по соединению (A, B). В данном случае все коммутаторы могут быть первого типа.

Рассмотрим теперь пример более сложных входных данных, представленный на рис. 3. Параллельное вычисление (рис. 3а) включает 10 процессов с требованиями по производительности 10 единиц. Для всех потоков данных, показанных на рисунке, требуется пропускная способность 40 единиц. Распределенная система (рис. 3b) содержит 10 вычислительных узлов h_0, ..., h_9 производительностью по 10 единиц, и коммутаторы A, B, C, D, E, F. Пропускная способность соединений, связывающих вычислительные узлы с коммутаторами – 100 единиц, соединения между коммутаторами имеют пропускную способность 50 единиц.

Первая ЦЛП-задача при назначении процессов P_0, ..., P_9 на узлы h_0, ..., h_9 соответственно решения не имеет. В данном случае причину уста-

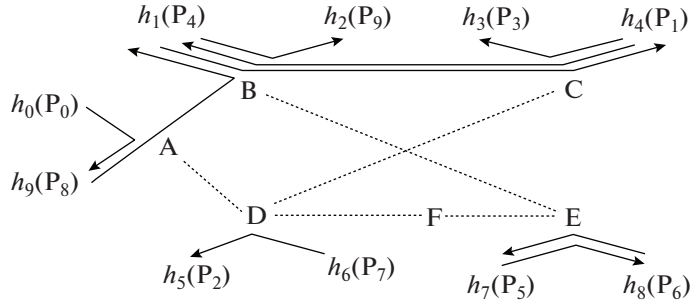


Рис. 4. Пример 2, решение второй ЦЛП-задачи.

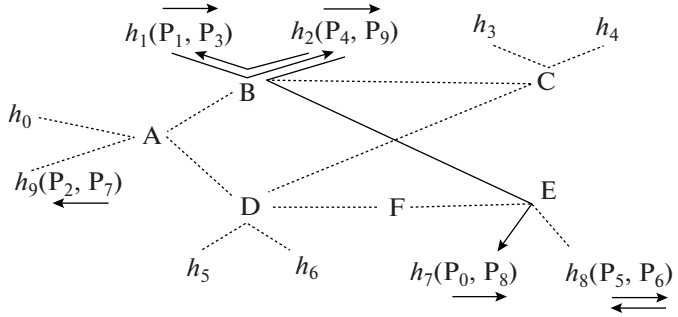


Рис. 5. Пример 2, решение второй ЦЛП-задачи при удвоенной производительности вычислительных узлов.

новить не сложно: пропускная способность соединений, связывающих коммутатор C с другими коммутаторами, недостаточна для трех входных потоков процессов P_3, P_4. Задача будет иметь решение, если добавить в систему (рис. 3б) одно из соединений (C, E) или (C, F).

Решение второй ЦЛП-задачи для тех же входных данных, в предположении, что все коммутаторы относятся к первому типу, изображено на рис. 4 (пунктиром показаны неиспользованные соединения).

Видно, что взаимодействующие процессы по возможности размещаются на смежных узлах, подключенных к одному коммутатору.

Если увеличить производительность вычислительных узлов до 20 единиц, то процессы будут сгруппированы на вычислительных узлах парами (рис. 5). Тем самым минимизируется максимальная и суммарная длина маршрутов, так как для процессов на одном узле длина маршрута нулевая.

При дальнейшем увеличении производительности узлов (30, 40, 50 единиц) происходит группирование до 3, 4, 5 процессов на узле, и происходит дальнейшее уменьшение суммарной длины маршрутов. При увеличении производительности вычислительных узлов до 60 единиц на одном из узлов размещается наибольшая слабая компонента связности графа (рис. 3а): P_0, P_1, P_3, P_4, P_8, P_9, а остальные процессы размещаются на другом узле; при этом суммарная длина маршрутов оказывается нулевой.

6. ЗАКЛЮЧЕНИЕ

Предложенный в работе подход может значительно упростить процедуру отображения сложных параллельных вычислений на распределенные системы, использующие коммуникационную среду RapidIO. Экспериментальный подбор требуемого отображения весьма трудоемок и не всегда обеспечивает оптимальный результат. Применение методов ЦЛП или других подобных методов гарантирует нахождение отображения, если оно существует, и позволяет оптимизировать его по различным параметрам.

Тем не менее, в этой области остается еще множество нерешенных вопросов, требующих дальнейших исследований. К их числу относятся вопросы масштабируемости предложенных формулировок ЦЛП-задач, дополнительные параметры оптимизации искомого отображения, более реалистичные модели параллельного вычисления и распределенной системы, наконец, задача построения распределенной системы по заданному параллельному вычислению. Остановимся на некоторых из них, представляющихся наиболее значимыми с практической точки зрения.

В целевой функции (11) предусмотрена минимизация максимальной длины маршрута, суммарной длины всех маршрутов и общее число записей в таблицах маршрутизации. К этому можно добавить:

- максимизацию минимального запаса пропускной способности по всем соединениям (чтобы избежать ситуаций, когда одни соединения используются на 100%, а другие не загружены);
- максимизацию минимального запаса производительности по всем вычислительным узлам;
- минимизацию максимальной нагрузки на коммутатор.

В формулировку задачи необходимо включить также ограничение на длину маршрутов, следующее из специфики коммуникационной среды RapidIO.

Для процессов, входящих в состав параллельных вычислений, может требоваться не только определенный уровень производительности, но и другие характеристики вычислительного узла, например, наличие определенных сопроцессоров. Кроме того, некоторые процессы должны получать входные данные от внешних устройств, поэтому они должны выполняться на вычислительных узлах, имеющих подключение к таким устройствам. Модели параллельного вычисления и распределенной системы, а также формулировку 2-й ЦЛП-задачи нетрудно расширить для учета подобных требований.

Еще один важный с практической точки зрения вопрос – ситуация, когда решение ЦЛП-задачи не существует. Причиной может быть недостаточное число соединений или их недостаточная пропускная способность, либо недостаточная производительность вычислительных узлов. Для того чтобы попытаться определить узкие места распределенной системы, можно рассмотреть решение модифицированной ЦЛП-задачи. В модифицированной задаче не используются ограничения по пропускной способности соединений и производительности узлов, но в целевой функции предусматривается минимизация максимального недостатка пропускной способности по всем соединениям и максимального недостатка производительности по всем вычислительным узлам. Рассмотрение полученного решения может дать подсказки о том, где именно следует добавить сетевые соединения, вычислительные узлы, коммутаторы.

Более радикальным решением была бы постановка задачи о построении распределенной системы под заданное параллельное вычисление, как в [23].

Пусть задано параллельное вычисление, а вместо распределенной системы определен набор

вычислительных узлов и набор коммутаторов. Для вычислительных узлов заданы уровни производительности (и, возможно, другие характеристики), а для каждого коммутатора – его тип и количество портов. Для портов заданы ограничения пропускной способности. Требуется построить:

1) множество соединений между вычислительными узлами и коммутаторами, а также между разными коммутаторами, где пропускная способность соединения определяется как минимум пропускных способностей портов, которые оно соединяет;

2) отображение процессов параллельного вычисления на вычислительные узлы полученной распределенной системы;

3) схему и таблицы маршрутизации, обеспечивающие потоки данных параллельного вычисления.

На практике распределенная система создается не из отдельных вычислительных узлов и коммутаторов, а из плат. На плате может быть расположено ограниченное количество микросхем (процессоров, микросхем памяти, коммутаторов) и разъемов для соединения с другими платами. Поэтому более реалистичной является постановка задачи, в которой заданы не отдельные вычислительные узлы и коммутаторы, а набор однотипных плат либо плат из ограниченного набора известных типов. В отличие от приведенной выше постановки задачи, требуется определить соединения не между отдельными узлами и коммутаторами, а между имеющимися платами.

Методы, подобные предложенному в работе, позволяют также решать некоторые вопросы отказоустойчивости. Например, для заданной распределенной системы и заданного параллельного вычисления (п. 4.2) можно проверить возможность альтернативного отображения параллельного вычисления при всех вариантах выхода из строя одного из элементов оборудования. Более сложная постановка задачи – построение отказоустойчивой распределенной системы под заданное параллельное вычисление, допускающей альтернативное отображение при одиночных отказах оборудования.

Таким образом, представленный в работе подход открывает возможности для целого круга исследований в области отображения параллельных вычислений на распределенные системы, а также построения распределенных систем, адаптированных к заданному параллельному вычислению.

7. БЛАГОДАРНОСТЬ

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН (проведение фундаментальных научных исследований 47 ГП) по теме № 0065-2019-0002 “Исследование и реализация программной платформы для перспективных многоядерных процессоров” (рег. № АААА-А19-119012290074-2).

СПИСОК ЛИТЕРАТУРЫ

1. RapidIO Interconnect Specification (Revision 1.3). http://www.rapidio.org/rapidio-specifications.
2. *Бобков С.Г., Задябин С.О.* Перспективные высокопроизводительные вычислительные системы промышленного применения на базе стандарта RapidIO // Электроника, микро- и наноэлектроника: сб. науч. тр. 11-й Рос. науч.-технич. конф. под ред. В.Я. Стенина. М.: Изд-во МИФИ, 2009. С. 114–121.
3. *Райко Г.О., Павловский Ю.А., Мелькановић В.С.* Технология программирования многопроцессорной обработки гидроакустических сигналов на вычислительных устройствах семейства “КОМДИВ” // Научно-технический сборник Гидроакустика. СПб.: ОАО “Концерн “Океанприбор”. 2014. Вып. 20 (2). С. 85–92.
4. *Сударева О.Ю.* Реализация алгоритма MG пакета NPB для многопроцессорного вычислительного комплекса на базе микропроцессора КОМДИВ128-РИО // Труды НИИСИ РАН. 2015. Т. 5. № 1. С. 75–87.
5. *Годунов А.Н., Солдатов В.А.* Конфигурирование многопроцессорных систем в операционной системе реального времени Багет // Программная инженерия. 2016. Т. 7. № 6. С. 243–251.
6. *Годунов А.Н., Солдатов В.А., Хоменков И.И.* Передача сообщений в коммуникационной среде RapidIO для семейства операционных систем реального времени Багет // Программная инженерия. 2020. Т. 11. № 1. С. 26–33.
7. Библиотека параллельной обработки сигналов // Труды НИИСИ РАН. 2015. Т. 5. № 1. С. 64–69.
8. *Грингауз Т.К., Онин А.Н.* Инструментальное программное обеспечение для подготовки запуска задач в мультипроцессорных комплексах реального времени // Труды научно-исследовательского института системных исследований Российской академии наук. 2015. Т. 5. № 2. С. 122–129.
9. *Павлов А.Н.* Инструментальный комплекс “РИО-оптимизатор” для разработки прикладного ПО с использованием Библиотеки параллельной обработки сигналов // Труды НИИСИ РАН. 2015. Т. 5. № 1. С. 70–74.
10. *Hoefler T., Jeannot E., Mercier G.* An overview of process mapping techniques and algorithms in high-performance computing. Wiley, 2014. *Jeannot E., Zilinskas J.* (eds.), High Performance Computing on Complex Environments. P. 75–94.
11. *Liu W., Yang L., Jiang W., Feng L., Guan N., Zhang W., Dutt N.* Thermal-Aware Task Mapping on Dynamically Reconfigurable Network-on-Chip Based Multiprocessor System-on-Chip // IEEE Transactions on Computers. 2018. V. 67. № 12. P. 1818–1834.
12. *Derin O., Kabakci D., Fiorin L.* Online task remapping strategies for fault-tolerant Network-on-Chip multiprocessors // Proceedings of the Fifth ACM/IEEE International Symposium, Pittsburgh, PA. 2011. P. 129–136.

13. *Huang K., Jiang X., Zhang X., Xiong D., Yan X.* Energy-Efficient Fault-Tolerant Mapping and Scheduling on Heterogeneous Multiprocessor Real-Time Systems // IEEE Access. 2018. V. 6. P. 57614–57630.
14. *Yang L., Liu W., Jiang W., Li M., Yi J., Sha E.H.* Application Mapping and Scheduling for Network-on-Chip-Based Multiprocessor System-on-Chip With Fine-Grain Communication Optimization // IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2016. V. 24. № 10. P. 3027–3040.
15. *Tang Q., Zhu L.-H., Zhou L., Xiong J., Wei J.-B.* Scheduling directed acyclic graphs with optimal duplication strategy on homogeneous multiprocessor systems // Journal of Parallel and Distributed Computing. 2020. V. 138. P. 115–127.
16. *Tang Q., Wu S., Shi J., Wei J.* Optimization of Duplication-Based Schedules on Network-on-Chip Based Multi-Processor System-on-Chips // IEEE Transactions on Parallel and Distributed Systems. 2017. V. 28. № 3. P. 826–837.
17. *Yu Y., Prasanna V.K.* Energy-balanced task allocation for collaborative processing in networked embedded systems // ACM SIGPLAN Notices. 2003. V. 38. № 7. P. 265–274.
18. *Liu W., Yi J., Li M., Chen P., Yang L.* Energy-Efficient Application Mapping and Scheduling for Lifetime Guaranteed MPSoCs // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2019. V. 38. № 1. P. 1–14.
19. *Huang K., Zhang X., Zheng D., Yu M., Jiang X., Yan X., de Brisolara L.B., Jerraya A.A.* A Scalable and Adaptable ILP-Based Approach for Task Mapping on MP-SoC Considering Load Balance and Communication Optimization // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2019. V. 38. № 9. P. 1744–1757.
20. *Kovalov A., Lobe E., Gerndt A., Lüdtke D.* Task-Node Mapping in an Arbitrary Computer Network Using SMT Solver // International Conference on Integrated Formal Methods IFM. 2017. Integrated Formal Methods. P. 177–191.
21. *Бакулин А.А.* Проверка допустимости схемы маршрутизации в системе RapidIO // Программные продукты и системы. 2011. № 4. С. 20–23.
22. GNU Linear Programming Kit. https://www.gnu.org/software/glpk.
23. *Bobda C., Ishebabi H., Mahr P., Mbongue J.M., Saha S.K.* MeXT: A Flow for Multiprocessor Exploration // IEEE High Performance Extreme Computing Conference (HPEC). 2019. P. 1–7.

ПРОГРАММИРОВАНИЕ, 2020, № 6, с. 67–72

ТЕОРЕТИЧЕСКИЕ ВОПРОСЫ ПРОГРАММИРОВАНИЯ

УДК 681.3.06

МОДЕЛИРОВАНИЕ МОГОЛЕНТОЧНЫХ МАШИН МИНСКОГО И ТЬЮРИНГА ТРЕХЛЕНТОЧНЫМИ МАШИНАМИ МИНСКОГО

© 2020 г. С. С. Марченков[a],*, С. Д. Макеев[a],**

[a] *МГУ им. М.В. Ломоносова, факультет вычислительной математики и кибернетики, ГСП-1, Ленинские горы, МГУ, д. 1, стр. 52, 119991 Москва, Россия*

**E-mail: ssmarchen@yandex.ru*

***E-mail: sergeymak98@gmail.com*

Поступила в редакцию 24.04.2020 г.
После доработки 06.05.2020 г.
Принята к публикации 17.06.2020 г.

Показано, что k-ленточную машину Минского, работающую с временем $T(n)$, можно промоделировать трехленточной машиной Минского за время, по порядку не превосходящее $T(n)^k \times \log T(n)$. Установлено, что многоленточные машины Тьюринга можно промоделировать трехленточными машинами Минского при “оптимальном” кодировании слов, записанных на лентах.

DOI: 10.31857/S0132347420060059

1. ВВЕДЕНИЕ

В теории алгоритмов широко используется моделирование одних вычислительных устройств другими вычислительными устройствами. Цели моделирования могут быть различными: сравнение вычислительных возможностей рассматриваемых устройств, доказательство существования устройств, обладающих теми или иными свойствами внешней или внутренней памяти, построение устройств с минимально возможными параметрами вычислений и другие.

Некоторые абстрактные вычислительные устройства (часто называемые машинами) в вопросах моделирования принято считать “эталонами”. К ним относятся машины Тьюринга различных типов, машины с произвольным доступом к памяти и машины Минского (см., например, [1–3]). Среди вычислительных устройств, имеющих внешнюю память в виде лент, многоленточные машины Тьюринга обладают, по-видимому, наиболее широкими вычислительными возможностями – в значительной степени за счет организации внешней памяти. Поэтому они довольно часто применяются для формализации различных алгоритмических процессов.

Машины Минского [2, 4, 5] представляют собой универсальные вычислительные устройства, способные вычислять произвольные частично-рекурсивные функции. В теории рекурсивных функций они являются весьма удобным средством для получения результатов сложностного и алгебраического характера [6–10]. Однако, в отличие от многоленточных машин Тьюринга, машины Минского обладают сравнительно небольшими возможностями по части хранения и извлечения информации из внешней памяти.

Одной из сложностных характеристик машины Минского может служить число имеющихся у нее лент (регистров). Известно [2, 4, 5], что на машинах с тремя лентами можно вычислять любые одноместные частично-рекурсивные функции (при стандартной записи аргумента на ленте). Для двухленточных машин это тоже возможно, однако аргумент и значение функции при этом следует кодировать экспоненциальным кодом. На одноленточных машинах Минского можно вычислить только некоторые тривиальные функции.

Таким образом, трехленточные машины Минского по ряду причин представляются одним из “минимальных” эталонных вычислительных устройств, которые в вычислительном отношении интересно использовать для моделирования других вычислительных устройств. Прежде всего, задача о моделировании возникает для самих многоленточных машин Минского, а также для многоленточных машин Тьюринга, вычисляющих одноместные функции. Основная проблема здесь заключается в том, чтобы найти “оптимальный” способ кодирования информации, имеющейся на лентах машины Минского или машины Тьюринга, который будет

доступен для обработки на трехленточных машинах Минского, и оценить время моделирования для такого способа кодирования.

В настоящей работе мы решаем сформулированную задачу: предлагаем некоторый “оптимальный” способ кодирования чисел, записанных на лентах многоленточной машины Минского или многоленточной машины Тьюринга; показываем, как для данного способа можно промоделировать оба вычислительных устройства трехленточными машинами Минского с “минимально возможными” потерями во времени. Для машин Минского этот переход выполняется с полиномиальным замедлением. Отметим, что несмотря на большое число работ, где одни вычислительные устройства моделируются другими вычислительными устройствами, моделирование универсальных вычислительных устройств трехленточными машинами Минского с оценками времени моделирования, насколько нам известно, не проводилось. Полученные в работе результаты можно будет использовать как при моделировании других вычислительных устройств (трехленточными машинами Минского), так и при решении специальных задач, относящихся, например, к вычислениям с полиномиальным временем.

2. ОСНОВНЫЕ ПОНЯТИЯ

Пусть $\mathbb{N} = \{0,1,\ldots\}$. Говоря далее о (натуральных) числах, мы имеем в виду числа из множества $\mathbb{N}$.

Будем рассматривать следующий вариант k-ленточной машины Тьюринга ($k \geq 1$). Машина Тьюринга $\mathcal{T}$ имеет k бесконечных в обе стороны лент, разбитых на клетки. На каждой ленте расположена одна считывающе-записывающая головка. Ленточный алфавит машины $\mathcal{T}$ состоит из символов 0, 1, λ (λ – “пустой” символ). Машина имеет множество (внутренних) состояний $Q = \{q_0, q_1, \ldots, q_r\}$, где q_1 – начальное, а q_0 – заключительное состояния. Функционирование машины $\mathcal{T}$ определяется программой, которая состоит из команд вида

$$a_1 \ldots a_k q_i \to b_1 \ldots b_k D_1 \ldots D_k q_j,$$

где $a_1, \ldots, a_k, b_1, \ldots, b_k \in \{0,1,\lambda\}$, $i \neq 0$ и $D_1, \ldots, D_k$ – символы движения на лентах: L, R, S. Предполагаем, что в программе машины $\mathcal{T}$ нет двух различных команд с одинаковыми левыми частями.

Нас будут интересовать функции одного аргумента $f(n)$, вычислимые на машине $\mathcal{T}$. Предполагаем, что в начальный момент времени на первой ленте машины записано двоичное представление числа n (слева и справа от этой записи лента пустая – заполнена символами λ), все остальные ленты пустые, машина находится в состоянии q_1, а ее головка на первой ленте – в клетке, которая раположена непосредственно справа от клетки, содержащей младший разряд двоичного представления n. Удобно также считать, что в заключительный момент времени результат $f(n)$ в двоичной записи представлен на первой ленте, слева от записи $f(n)$ лента пустая, а головка машины $\mathcal{T}$ на первой ленте находится в (пустой) клетке, примыкающей справа к записи $f(n)$.

Машина Минского представляет собой вариант нестирающей многоленточной машины Тьюринга. Машина Минского $\mathcal{M}$ имеет k односторонних (бесконечных вправо) лент, содержимое которых не меняется в процессе вычисления, и множество состояний $Q = \{q_0, q_1, \ldots, q_r\}$. Концевые клетки лент содержат символ 1, все остальные клетки – символ 0. На каждой из лент находится одна читающая головка. В процессе работы машины $\mathcal{M}$ на каждом шаге вычисления любая из головок может независимым образом сдвигаться на одну клетку влево, вправо или оставаться в той же клетке. Функционирование машины $\mathcal{M}$ определяется программой, которая состоит из команд вида

$$a_1 \ldots a_k q_i \to q_s d_1 \ldots d_k,$$

где $a_1, \ldots, a_k$ – символы 0 или 1, обозреваемые головками машины $\mathcal{M}$ на лентах, $1 \leq i \leq r$ и $d_1, \ldots, d_k$ – символы движения головок на лентах ($d_1, \ldots, d_k \in \{-1, 0, 1\}$). Программа машины $\mathcal{M}$ организована таким образом, что головки не могут сходить с концевых клеток лент.

Машина $\mathcal{M}$ вычисляет (частичную) функцию $f(n)$, если в начальный момент вычисления головка на первой ленте находится в клетке с номером n (концевые клетки имеют номер 0), остальные головки – в клетках с номером 0, а машина – в начальном состоянии q_1. Если значение $f(n)$ определено, то машина через конечное число тактов достигает заключительного состояния q_0, при этом в заключительный момент времени первая головка находится в клетке с номером $f(n)$. Если же значение $f(n)$ не определено, то машина работает неограниченно долго.

3. РЕЗУЛЬТАТЫ

При моделировании одного вычислительного устройства $\mathcal{M}_1$ другим вычислительным устройством $\mathcal{M}_2$ процесс моделирования, как правило, разбивается на три этапа. На первом этапе исход-

ные данные, записанные в языке устройства $\mathcal{M}_1$, переводятся в язык устройства $\mathcal{M}_2$ (будем говорить о кодировании данных). Это может быть, например, переход в другую позиционную систему счисления либо переход к коду, который учитывает определенные параметры моделируемого устройства $\mathcal{M}_1$.

На втором этапе устройство $\mathcal{M}_2$, работая в кодах, по шагам моделирует процесс применения устройства $\mathcal{M}_1$ к входным данным (собственно моделирование). На третьем этапе происходит перевод полученного результата из языка устройства $\mathcal{M}_2$ в язык устройства $\mathcal{M}_1$ (коротко: декодирование). Каждый из этапов имеет свои особенности. Однако первый и третий этапы идейно близки друг к другу и выполняются несколько проще, чем второй этап.

В основе моделирования машин Минского (ММ) и машин Тьюринга (МТ) трехленточными машинами Минского лежит некоторый способ кодирования наборов натуральных чисел. Этот способ не новый, он использовался другими авторами примерно для аналогичных целей (см., например, [11] или [12]).

Пусть $d, n_1, \ldots, n_k$ – натуральные числа, $d \geq 2$ и $a_{1m} \ldots a_{10}$, ..., $a_{km} \ldots a_{k0}$ – представления чисел $n_1, \ldots, n_k$ в системе счисления с основанием d (если длины d-ичных представлений некоторых чисел n_j меньше $m+1$, то соответствующие старшие разряды представлений считаем равными нулю). Набору чисел $(n_1, \ldots, n_k)$ (а точнее, набору d-ичных представлений этих чисел) сопоставим число Code $(n_1, \ldots, n_k)$ с d-ичным представлением

$$a_{km}a_{k-1,m} \ldots a_{1m}a_{k,m-1}a_{k-1,m-1} \ldots a_{1,m-1} \ldots a_{k0}a_{k-1,0} \ldots a_{10}. \quad (1)$$

Как видно, в представлении (1) разряды числа n_1 занимают позиции с номерами $1, k+1, 2k+1, \ldots$, разряды числа n_2 – позиции с номерами $2, k+2, 2k+2, \ldots$ и т.д. Мы будем говорить об i-й "дорожке", занимаемой разрядами числа n_i.

Не вдаваясь в детали, опишем в общих чертах, как трехленточная ММ может совершать некоторые простые действия с числом $n = \text{Code}(n_1, \ldots, n_k)$, опираясь при этом на представление числа n в виде (1). Будем предполагать известным, как на трехленточных ММ можно сравнивать имеющиеся на лентах числа с нулем и вычислять арифметические функции $x+d$, $x-d$ (d – натуральное число, а в случае вычитания выполняется неравенство $x \geq d$), $d \times x$, $[x/d]$, $\text{rm}(x,d)$ ($d \geq 2$ и $\text{rm}(x,d)$ и обозначает остаток от деления x на d).

Определение разряда a_{i0} $(1 \leq i \leq k)$. Число n делим с остатком на d, получаем частное $[n/d]$, которое записываем на вторую ленту, и остаток a_{10}. Если $i = 1$, умножаем $[n/d]$ на d и добавляем к результату a_{10}. Задача решена.

В противном случае записываем число 1 на третью ленту (это делается для того, чтобы далее не "потерять" возможные нулевые разряды $a_{10}, a_{20}, \ldots$), умножаем 1 на d и добавляем к результату найденное число a_{10}. Если уже проделано $j < i$ шагов и на первой ленте образовалось число n', d-ичное представление которого получается из представления (1) удалением последних j разрядов, а на третьей ленте – число n'' с d-ичным представлением $1a_{10} \ldots a_{j0}$, то делим число n' на d с остатком $a_{j+1,0}$, умножаем число n'' на d и добавляем к полученному результату $a_{j+1,0}$. Если $j+1 = i$, то искомый разряд найден. Далее меняем ролями числа n' и n'', дописывая к представлению числа n' удаленные разряды $1a_{10} \ldots a_{j0}$ и добавленный разряд 1, в конце процедуры вычитаем из полученного числа 1 (удаление "лишнего" разряда 1). Если же $j+1 < i$, то продолжаем описанную выше процедуру до вычисления разряда a_{i0}.

Обращение представления (1). Задача состоит в том, чтобы по исходному числу n с d-ичным представлением (1) получить число n', d-ичное представление которого представляет собой обращение представления (1) с добавленной в начало единицей (о роли единицы сказано выше). По существу алгоритм построения числа n' указан выше: необходимо сначала записать на третью ленту единицу, затем умножить ее на d, путем деления числа n на d с остатком, найти разряд a_{10}, добавить его на третью ленту, умножить полученный результат на d и т.д. до тех пор, пока очередное частное от деления не станет равным нулю.

Нетрудно видеть, что выполнение обращения на трехленточной ММ требует по порядку не более $n \times \log n$ шагов: при вычислении очередного разряда представления (1) машине необходимо разделить на d с остатком одно число, не превосходящее n, и умножить на d другое число, не превосходящще n. Поскольку количество разрядов в представлении (1) по порядку равно $\log n$, приходим к верхней оценке времени обращения, по порядку равной $n \times \log n$.

Перейдем к более сложным действиям, выполнимым на трехленточных ММ.

Вычитание единицы (из числа n_i в представлении (1) числа Code($n_1, \ldots, n_k$)). Предполагаем, что в начальный момент времени на первой ленте трехленточной ММ находится число $n = \text{Code}(n_1, \ldots, n_k)$ и $n_i > 0$. Выполняемая задача состоит в том, чтобы на i-й дорожке представления (1) числа n найти первый ненулевой разряд, вычесть из него 1, а все предыдущие нулевые разряды (на этой дорожке) заменить числом $d-1$. Так же, как в задаче обращения (с добавлением единицы в начало обращения), ММ путем последовательного "перекладывания" разрядов представления (1) с первой ленты на третью начинает разыскивать на i-й дорожке первый ненулевой разряд, одновременно заменяя нулевые разряды числом $d-1$. Ввиду условия $n_i > 0$ такой разряд непременно найдется. Чтобы определить, что он находится на i-й дорожке, машине необходимо выполнять действия "по модулю" k: первый разряд на i-й дорожке появится при "обработке" i-го разряда представления (1), второй разряд – при обработке $(k+i)$-го и т.д. Как только нужный разряд будет найден, машина заменит его разрядом, на единицу меньшим, а далее, как в задаче обращения, будет перекладывать разряды представления числа с третьей ленты на первую ленту. В заключение машина вычтет из полученного числа 1.

Поскольку при выполнении операции *вычитание единицы* машине в "самом худшем" случае придется обработать порядка $\log n$ разрядов представления (1), общее время выполнения данной операции по порядку не превосходит величины $n \times \log n$.

Прибавление единицы (к числу n_i в представлении (1) числа Code ($n_1, \ldots, n_k$)). Эта задача является обратной к предыдущей задаче. Поэтому остановимся лишь на некоторых отличительных особенностях. Если среди разрядов числа n_i есть хотя бы один, отличный от $d-1$, то реализуем процедуру, обратную к процедуре *вычитание единицы*. Именно, если разряды $a_{i0}, \ldots, a_{ij}$ равны $d-1$ и $a_{i,j+1} \neq d-1$, то заменяем разряды $a_{i0}, \ldots, a_{ij}$ нулями, а разряд $a_{i,j+1}$ – числом $a_{i,j+1}+1$. Так же поступаем, когда $a_{im} = 0$. Однако если $a_{im} = d-1$, то необходимо добавить нулевые разряды $a_{1,m+1}, \ldots, a_{i-1,m+1}$ и единичный разряд $a_{i,m+1}$.

Удаление младшего разряда (из представления числа n_i в представлении (1) числа Code ($n_1, \ldots, n_k$)). Задача состоит в том, чтобы в представлении (1) разряды $a_{im}, a_{i,m-1}, \ldots, a_{i1}, a_{i0}$ заменить разрядами 0, $a_{im}, \ldots, a_{i2}, a_{i1}$. Задача решается по аналогии с предыдущими задачами, однако теперь основные действия проводятся после того, как на третьей ленте будет получено обращение представления (1). В процессе "перекладывания" разрядов с третьей ленты на первую необходимо разряд a_{im} заменить нулем и запомнить. Через k позиций разряд $a_{i,m-1}$ заменяется разрядом a_{im} и запоминается до момента обработки разряда $a_{i,m-2}$ и т.д. до разряда a_{i0}, который заменяется разрядом a_{i1}.

Добавление младшего разряда (к представлению числа n_i в представлении (1) числа Code ($n_1, \ldots, n_k$)). Хотя эта задача по форме выглядит как обратная к задаче *уаление младшего разряда*, выполняется она несколько иначе. Именно необходимо сразу при получении обращения представления (1) поставить выбранное число a (котором машина хранит во внутренней памяти) вместо разряда a_{i0}, запомнить этот разряд, а далее последовательно заменять разряды $a_{i,j+1}$ разрядами a_{ij}. Исключение составляет случай, когда разряд a_{im} ненулевой. Тогда следует заменить его разрядом $a_{i,m-1}$ и затем (как в задаче *прибавление единицы*) добавить нулевые разряды $a_{1,m+1}, \ldots, a_{i-1,m+1}$ и ненулевой разряд $a_{i,m+1}$, равный a_{im}.

Так же, как при рассмотрении операции *вычитание единицы*, каждую из последних трех операций можно выполнить на трехленточной ММ за время, по порядку не превосходящее $n \times \log n$.

Теорема 1. *Пусть k-ленточная ММ работает с временем $T(n)$, причем $T(n) \geq n$. Тогда ее можно промоделировать трехленточной ММ, работающей с временем, по порядку не превосходящим $T(n)^k \times \log T(n)$.*

Доказательство. Пусть $a_m \ldots a_0$ – двоичное представление числа n. Тогда $n' = \text{Code}(n, 0, \ldots, 0)$ есть число с двоичным представлением

$$a_m 0^{k-1} a_{m-1} 0^{k-1} \ldots 0^{k-1} a_0, \qquad (2)$$

где 0^{k-1} – последовательность из $k-1$ нулей. На первом этапе моделирования трехленточная ММ $\mathcal{M}$ преобразует число n, записанное на ее первой ленте, в число n'.

Это делается примерно так же, как было описано выше при выполнении процедуры *обращение представления*. Именно сначала машина $\mathcal{M}$ с помощью последовательных делений числа n на 2 с остатком образует на третьей ленте число с двоичным представлнием

$$1 a_0 0^{k-1} a_1 0^{k-1} \ldots 0^{k-1} a_m, \qquad (3)$$

которое, за исключением единицы в начале, является обращением представления (2). Затем она (уже без измения разрядов двоичного представления) обращает представление (3) и вычитает из полученного числа 1.

Нетрудно видеть, что весь процесс получения числа Code $(n, 0, \ldots, 0)$ занимает на машине $\mathcal{M}$ время, по порядку не превосходящее n^k, т.е. не превосходящее $T(n)^k$.

Заметим, что для k-ленточной ММ, работающей с временем $T(n)$, в любой момент времени на любой из лент находится число, не превосходящее $T(n)$. Поэтому соответвующая величина Code $(n_1, \ldots, n_k)$ по порядку не превосходит $T(n)^k$. Обращаясь теперь к шагам моделирования, выполняемым машиной $\mathcal{M}$ на втором этапе, на основе указанных выше оценок для выполнения операций *вычитание единицы* и *прибавление единицы* приходим к заключению, что каждый шаг моделирования требует времени по порядку не более $T(n)^k \times \log T(n)$. Поэтому в целом весь второй этап займет по порядку не более $T(n)^{k+1} \times \log T(n)$ шагов.

Третий этап моделирования в значительной степени аналогичен первому этапу. Поэтому без подробных выкладок приводим верхнюю оценку (по порядку) его выполнения на машине $\mathcal{M}$: $T(n)^k \times \log T(n)$. Это завершает доказательство теоремы.

Ниже в теореме 2 мы предполагаем, что n есть величина аргумента (а не длина его двоичной записи).

Теорема 2. *Произвольную k-ленточную МТ, имеющую время работы $T(n)$, где $T(n) \geq n$, можно промоделировать трехленточной машиной Минского за время, по порядку не превосходящее $T(n)^2 \times 3^{2k \times T(n)}$.*

Доказательство. Пусть $\mathcal{T}$ — k-ленточная МТ с ленточным алфавитом $\{0, 1, \lambda\}$, которая работает с временем $T(n)$. Чтобы закодировать содержимое всех лент машины $\mathcal{T}$ одним числом, для любого i $(1 \leq i \leq k)$ разделим i-ю ленту машины $\mathcal{T}$ на две части: левую L_i, которая расположена непосредственно слева от клетки, обозреваемой в данный момент головкой машины, и правую R_i, крайнюю левую клетку которой в этот момент обозревает головка машины. Части L_i сопоставим число l_i, троичное представление которого находится на этой части. При этом считаем, что ленточным символам λ, 0, 1 соответствуют разряды 0, 1, 2 троичного представления l_i (предполагаем, что только конечное число клеток части L_i содержат символы 0 или 1). Аналогичное соглашение действует для части R_i и числа r_i, однако здесь по техническим причинам удобно считать, что младшие разряды части R_i располагаются слева. При этих соглашениях содержимое всех лент машины $\mathcal{T}$ будет кодироваться числом Code $(l_1, r_1, l_2, r_2, \ldots, l_k, r_k)$, в троичном представлении которого разряды числа l_i располагаются в позициях с номерами $2i - 1$, $2i + 2k - 1$, $2i + 4k - 1$, ... (дорожка с номером $2i - 1$), а разряды числа r_i — в позициях с номерами $2i$, $2i + 2k$, $2i + 4k$, ... (дорожка с номером $2i$).

Определим трехленточную ММ $\mathcal{M}$, которая будет моделировать машину $\mathcal{T}$. Первый этап работы машины $\mathcal{M}$ состоит в том, чтобы число n, записанное в начальный момент на первой ленте, перевести в число Code $(l, 0, 0, 0, \ldots, 0, 0)$, где троичная запись числа l получается из двоичной записи числа n заменой разрядов 0 и 1 соответственно разрядами 1 и 2. Это можно выполнить примерно так, как подобные процедуры были изложены выше. Именно необходимо разделить число n на 2 с остатком, к остатку добавить 1 и результат записать на третью ленту. Если $[n/2] = 0$, то машина $\mathcal{M}$ переносит результат с третьей ленты на первую, и первый этап моделирования завершен. В противном случае полученное на третьей ленте число умножается на 3^{2k-1}. Тем самым машина $\mathcal{M}$ выполнила "обработку" первого разряда двоичного представления числа n (умножение на 3^{2k-1} соответствует в данном случае постановке $2k - 1$ нулей в качестве первых разрядов чисел $r_1, l_2, r_2, \ldots, l_k, r_k$).

Далее машина $\mathcal{M}$ переходит к аналогичной обработке второго разряда двоичного представления n: делит число $[n/2]$ на 2 с остатком, умножает число с третьей ленты на 3, добавляет к нему остаток от деления плюс 1, умножает полученный результат на 3^{2k-1} и так далее. После завершения обработки последнего разряда двоичного представления n машина $\mathcal{M}$ осуществляет, как это делалось выше, преобразование полученного числа в число с "обращенным" троичным представлением.

Нетрудно видеть, что машина $\mathcal{M}$ затратит на выполнение первого этапа моделирования по порядку не более $3^{2k \cdot \log_2 n} \times \log n$ тактов.

Теперь машина $\mathcal{M}$ будет моделировать каждый шаг работы машины $\mathcal{T}$. Чтобы это выполнить, машине $\mathcal{M}$ необходимо определить все младшие разряды в троичных представлениях чисел $r_1, \ldots, r_k$. Это делается на основе процедуры *определение разряда a_{i0}*, описанной выше. Вычислив эти раз-

ряды, машина $\mathcal{M}$ определит команду машины $\mathcal{T}$, выполняемую в данный момент ("текущее" состояние машины $\mathcal{T}$ машина $\mathcal{M}$ хранит во внутренней памяти). Теперь машине $\mathcal{M}$ необходимо выполнить требуемые преобразования: замена младших разрядов в представлениях чисел $r_1,\ldots,r_k$, удаление младшего разряда из представления числа l_i и добавление младшего разряда к представлению числа r_i (если машина $\mathcal{T}$ сдвигает i-ю головку влево) и, наоборот, удаление младшего разряда из представления числа r_i и добавление младшего разряда к представлению числа l_i (если машина $\mathcal{T}$ сдвигает i-ю головку вправо).

Замена младших разрядов выполняется так же, как процедура *определение разряда* a_{i0}, описанная выше. Процедуры удаления и добавления младших разрядов также приведены выше. Нетрудно видеть, что все три процедуры машина $\mathcal{M}$ может выполнить за время, по порядку не превосходящее величины $n\times\log n$, где n – код набора $(l_1,r_1,l_2,r_2,\ldots,l_k,r_k)$.

Данная оценка времени моделирования одного шага работы машины $\mathcal{T}$ позволяет оценить в целом все время, затрачиваемое машиной $\mathcal{M}$ на втором этапе моделирования. В самом деле, если машина $\mathcal{T}$ работает с временем $T(n)$, то в течение всего вычисления длина троичной записи на любой из "полулент" также не превосходит $T(n)$. Отсюда вытекает верхняя оценка $3^{T(n)}$ для чисел, записанных на этих "полулентах". Следовательно, в процессе вычисления величина n по порядку будет не больше, чем $3^{2k\times T(n)}$. Учитывая полученную выше оценку $n\times\log n$ времени моделирования одного шага работы машины $\mathcal{T}$ и общее время $T(n)$ ее работы, приходим к верхней оценке (по порядку) вида $T(n)^2\times 3^{2k\cdot T(n)}$.

Для завершения доказательства теоремы осталось рассмотреть третий этап моделирования машины $\mathcal{T}$. По сути он аналогичен первому этапу моделирования, однако здесь надо учесть, что в начале третьего этапа на "входе" машины $\mathcal{M}$ имеется величина порядка $3^{2k\cdot T(n)}$. Так же, как на первом этапе моделирования, время работы машины $\mathcal{M}$ можно ограничить по порядку величиной $3^{2k\times T(n)}$, умноженной на логарифм этой величины, что по порядку не превосходит величины $T(n)^2\times 3^{2k\times T(n)}$.

4. БЛАГОДАРНОСТИ

Работа выполнена при частичной поддержке Российского фонда фундаментальных исследований (проект 19-01-00200).

СПИСОК ЛИТЕРАТУРЫ

1. *Марченков С.С., Савицкий И.В.* Машины в теории вычислимых функций. М.: МАКС Пресс, 2018.
2. *Мальцев А.И.* Алгоритмы и рекурсивные функции. М.: Наука, 1986.
3. *Clote P.* Computation models and functional algebras // Handbook of Computability Theory. Elsevier Science B.V., 1999. P. 589–681.
4. *Minsky M.L.* Recursive unsolvability of Post's "TAG" and topics in theory of Turing machines // Ann. Math. 1961. V. 74. P. 437–455.
5. *Минский М.* Вычисления и автоматы. М.: Мир, 1971.
6. *Марченков С.С.* Базисы по суперпозиции в классах рекурсивных функций // Математические вопросы кибернетики. Вып. 3. 1991. С. 115–139.
7. *Волков С.А.* Пример простой квазиуниверсальной функции в классе $\mathscr{E}^2$ иерархии Гжегорчика // Дискретная математика. 2006. Т. 18. № 4. С. 31–44.
8. *Волков С.А.* Конечная порождаемость некоторых групп рекурсивных перестановок // Дискретная математика. 2008. Т. 20. № 4. С. 61–78.
9. *Марченков С.С.* О сложности класса $\mathscr{E}^2$ Гжегорчика // Дискретная математика. 2010. Т. 22. № 1. С. 5–16.
10. *Марченков С.С.* Представление функций суперпозициями. М.: КомКнига, 2010.
11. *Hennie F.C., Stearns R.E.* Two-tape simulation of multitape Turing machines // Journal of ACM. 1966. V. 13. № 4. P. 533–546.
12. *Яблонский С.В.* Введение в дискретную математику. М.: Наука, 1979.